UiO : Department of Informatics
University of Oslo

Performance analysis and dynamic reconfiguration of a SR-IOV enabled OpenStack cloud

Mohsen Ghaemi

Master's Thesis Spring 2014

Performance analysis and dynamic reconfiguration of a SR-IOV enabled OpenStack cloud

Mohsen Ghaemi

20th May 2014

Abstract

Cloud computing is growing very fast. Within a short time, it has become one of the mainstream trends in world of IT and many businesses intend to benefit from this technology. One of the main objectives of deploying clouds is to achieve higher efficiency. Increasing the efficiency is entirely dependent to improving the performance, and the total performance of a system is result of subsystems performance. Infrastructure as a Service (IaaS) is the most common model of cloud computing. This model not only is an independent service that is delivered to users but also serves the upper layers of clouds (PaaS and SaaS). A lot of studies in the area of cloud computing have focused on improving the performance of this layer. IaaS mainly consists of Networking and Virtualization where these two concepts are mutually interdependent. Working on improving each of them inevitably involves another. Single Root I/O Virtualization (SR-IOV) is an emerging technology aims to improve the I/O performance in virtual environments, while decreasing the virtualization overhead. It allows an I/O device to be shared by multiple Virtual Machines (VMs), without losing performance. This study has conducted comprehensive experiments to evaluate performance of Ethernet SR-IOV, compared it to other options, and investigated its impacts to the system. An OpenStack IaaS platform was configured to utilize the Ethernet SR-IOV interface and automatically assign Virtual Functions (VFs) to instances. Finally a prototype method was implemented to conduct live migration of SR-IOV attached VMs while maintaining the connectivity. The results show that SR-IOV can achieve a stable line rate throughput (more than 9 Gbps), increase the efficiency and scale network, without sacrificing live migration.

.

Dedicated to my supervisor

Vangelis Tasoulas

who was not only a supervisor during this project, but also a kind teacher who made me to learn so many things and a good friend who never gave up to support me.

Also I would like to
express my best and deepest gratitude to my beloved family including

My lovely mother, honorable father and dear brothers

for who I am and what I have now. I would achieve none of them without their total support. I would tell them *I Love You.*

Acknowledgments

It is an honour for me to express my appreciation to the following people and organisations, and recognise their supports:

- **University of Oslo** and **Oslo and Akershus University College** for offering this master degree program and providing a top quality study environment and facilities.
- **Simula Research Laboratory** for hosting this project and providing suitable research environment.
- **Hårek Haugerud**, who is really a gentleman and a great professor, for all the things I have learned from him and all of his sincere efforts to support us.
- **Kyrre Begnum**, who is one of the most knowledgeable and expert professors and sysadmins I have ever met, for all of his unique lectures and trainings as well as his supports and tips during this project.
- **Professor Paal E. Engelstad**, for giving me the opportunity of being his assistant and for his scientifical and emotional supports during this project.
- **Ismail Hassan** for putting us in challenging situations during his classes to teach us *how a sysadmin should struggle with new challenges.*
- **Professor Dag Langmyhr**, the academic head of NSA master program at UiO, for his comments to improve my thesis.
- **Laine Stump**, the senior software engineer at Red Hat, for his tips and supports about the bug of libvirt 1.1.1 .
- **Yoann Juet**, the head of IT security at the University of Nantes - France, for sharing his experience about SR-IOV interface issues and solutions.
- My dear brother **Ehsan Ghaemi**, for being my best teacher and guide during my entire life and for his scientifical supports during this project.
- **Kamyar Akhbari**, for being such a supportive, honest and reliable friend , and still keeping in touch even if the distance is long.
- **Sahar (Nasibeh) Tajik** , who has been a very nice friend with an endless grace and kindness, for her emotional supports during this project.
- My friends **Shahab Moghaddam** and **Farinaz Kahnamouie**, for their helps and supports to improve my thesis by reviewing my texts.
- My friends **Mohammadreza Ghasemi** and **Paria Tahai**, for their friendship and supports.
- And last but not least, All **Peer Students** in master program of **Network and System Administration** (Class of 2014), who really have been the best group of students ever. And I would like to take this opportunity to thank all of our norwegian friends in this group who really helped us feeling at home.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Motivation

Cloud computing is going to be the first choice for most of users and service providers in near future [23][50][59]. It is going to reshape many of IT processes in different businesses. Although this concept was very publicly considered around 2006 [59], just within six years it has become one of the mainstream trends in the world of IT. The main idea behind cloud computing is to provide computing resources in all levels of software, platform and infrastructure on demand [50, 85]. It can be known as the most promising and dynamic IT infrastructure technologies available to enterprises. A wide range from large multi-national enterprises to SMEs (Small to Medium sized Enterprises), all of which are able to enjoy the benefits delivered by the cloud computing paradigm[87]. A survey by Intel on 2012 turned out that almost 80% of 200 IT companies which surveyed already deployed or are deploying their private clouds [8] Another survey conducted by CloudPassage on 2013 showed that almost 94% of enterprises and 68% of SMBs (SMEs) that surveyed already deployed or are deploying their own clouds [13]. This transition from an uncertain high-tech concept to a commonly used model has been rapid . It is an indication that most of concepts and businesses models related to or dependent on IT in very near future will be designed based on cloud computing. This shows that many studies and researches are required in the area of cloud computing. While the cloud computing technology is still immature, it is time to work on different ideas and utilize brand new technologies which mutually contribute to development of cloud and cloud is a suitable platform to implement them.

Cloud computing is known in three forms of Public, Private and Hybrid cloud [26][3]. The term public cloud refers to a commercial cloud

which infrastructures of cloud are provided off-site and owned by third parties. These commercial clouds deliver their different services publicly via internet e.g. Amazon, Rackspace, etc. Private clouds are those which equipments and infrastructures of cloud (data center) are internal and often owned by the business. Hybrid cloud is a combination of two other types in some way. It could be an environment that the business provides and manages some resources in-house and has others provided externally. Universities, research centers and IT businesses are main deployers of private clouds. A private cloud can offer a good testbed for research or suitable platform and infrastructure for developing applications. There are some open source cloud solutions (also known as cloud management platforms or framework) such as Xen Cloud Platform (XCP), Eucalyptus, Openstack and Open Nebula [41] [83] [62] [22].

Openstack is one of the more recent projects which has drawn so much attention to itself. A very simple search of term "openstack" in google returns about 2.5 million results [1] on the other hand the same experiment with "opennebula" [2]or "Eucalyptus cloud" [3]returns even less search results. The project was initiated by NASA and Rackspace in 2010 and was adopted by Ubuntu Linux developers on 2011. Now more than 200 companies have joined the project and are in collaboration among which are AMD, Cisco, intel, HP, IBM,VMware, and Yahoo [24][56][64]. Openstack is a group of open source projects (softwares) that provides a cloud computing platform.

Core objectives in deploying clouds are on demand services, better performance and more efficiency[44]. Since clouds offer both hardware and softwares resources delivered as service, these matters are considered in all three layers of SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service) of cloud[50][85]. This leads to different availability, performance and efficiency considerations. Two top layers of SaaS and PaaS are laid on IaaS layer, therefore the QoS (Quality of Service) of whole cloud is strongly related to QoS of IaaS. The fact is that a cloud is a combination of several technologies among which virtualization and networking stand at the core. Therefore the focus of these considerations are on QoS of VMs and Networks.

[1]On March 10th,2014 using google.com returned 2,470,000 results
[2]On the same date same engin returned 975,000 results
[3]On the same date same engin returned 1,920,000 results

Furthermore, cloud computing is known as one of the promising strategies to achieve green computing[6][50]. Because of considerable increase in usage of ICT, there is a big concern about environmental impacts of this business. More computing nodes means more electricity consumption and more emission of greenhouse gases (GHG). Decreasing overheads leads to better performance and less utilization of resources. Achieving better performance on each compute node and the whole cloud brings more efficiency. Being more efficient in cloud environments means much more efficient use of data center equipments that leads to much less energy consumption and being greener. This matter not only leads to environmental care but also has economic benefits since the cost of energy is increasing everywhere.

Clouds heavily rely on virtualization technology because they utilize it on a large scale to support as many as possible virtual machines, virtual networks and other virtual resources on physical servers. Virtualization helps increase of efficiency by sharing physical resources between a number of VM Instances. Both of big providers which offer public clouds and enterprises that deployed their own private cloud, strongly tend to benefit from this technology as much as possible. Due to this fact sometimes a significant number of VMs are hosted on a physical server to optimize the utilization of data centers resources and thereby reducing the cost of energy and maintenance[43][81]. Since VM instances normally share physical processor and I/O interfaces this may impacts the computation and communication performance of cloud. likewise this matter may leads to loss of availability of services or security issues which are very critical[19].

In order to hesitate mentioned problems VM instances are migrated. Migration is the technique of moving VMs between physical hosts that moves entire OS and its associated application from one physical server to another. Live migration of VMs means doing the transfer in a way that the virtual machine stays responsive constantly during the process of migration. VMs are migrated by different techniques and with different aims such as power saving, load balancing, fault tolerance or maintenance[43][86]. Load balancing refers to distributing a heavy process or memory load on a specific physical server to others. Fault tolerance refers to immediate transfer of VMs from a physical machine which deals with a failure. It shows that live migration is a key characteristic of a cloud.

However there might be enough physical resources and migrating the VMs helps the load balancing and increase of performance, but efficiency also should be considered seriously. Energy saving is one of the most important points that almost all of providers are looking for. On the other hand it should be noticed that in many cases like SMBs infrastructure is limited. The better performance may be achieved by a good resource management not only by distributing loads along the physical resources. Since VMs do not have any physical limitations they are very flexible to manipulate their resources. Nowadays with very powerful processing units and dynamic resource allocation techniques such as dynamic memory allocation [75], main consideration of performance is about I/O and specifically networking[19] to provide high QoS.

The other axis of cloud is its network. Cloud computing can be seen as a kind of parallel and distributed system, and resources are shared through physical and virtual networks. VMs that are hosted in different compute nodes are managed and accessed through networks and communicate with each other through a path which may include a combination of virtual and physical networks[26]. It means that all services are accessed via network and this is common across all functions in a cloud. So the total QoS is strongly related to performance network hence without high performance networks developments in cloud services will not happen[54]. Virtual instances which have the main role in the cloud to deliver different services, use the physical Network Interface Card (NIC) of the host machine. However the technique of accessing to this I/O device would be different but finally all related traffics will pass through it. Networking in clouds should be reliable and stable. Most of cloud infrastructures are equipped with fast Ethernet one Gigabyte (1 Gb) NICs. It might be enough bandwidth for normal load of service requests. Due to recent advances in virtualization and multicore processors a considerable number of VMs can be hosted by a single physical machine. Number of VMs and their activities increases (or decreases) on demand and sometimes may not be predictable. Increase in the number of VMs may leads to have bottleneck in point of NIC. The other aspect of networking issue is emulating the network card (softwarebased I/O virtualization) when there are many VMs. IT will have a considerable process overhead since hypervisor should interfere all the time[30][89][19][48]. Moreover full virtualization or paravirtualization of a device will affect the performance of the device.

In Order to reduce the overhead of hypervisor intervention, some new techniques has been introduced[34][21]. Pass-through and Single Root I/O Virtualization (SR-IOV) are two techniques for this purpose. In these techniques a VM (guest) directly access the physical I/O device and there is no need to emulate it by hypervisor [30] [19] [89]. In pass-through technique a single device is assigned exclusively to a single VM directly. It helps to increase performance of the utilized device (near the native performance) and decreases the process overhead caused by device virtualization. But as in this technique only one VM has access to the device, due to limited number of I/Os (e.g. NIC) it will be against the scalability characteristic of virtualization. Furthermore direct assignment of physical I/O device to VM is an issue for live migration.

SR-IOV is a new technique introduced by the PCI-SIG organization, which offers a set of hardware enhancements to the PCIe device. This idea is retrieved from idea behind pass-through device but considers the sharing issue . SR-IOV technique aims to provide natively shareable devices. Although Implementing this technique requires utilizing specific hardware, not only improves performance but also provides resource sharing. a device which is SR-IOV-capable can create multiple instances of PCI function entities. These light-weight functions are called Virtual Functions (VFs). Each VF is supposed to be assigned to a single VM in order to direct access to device[73][34]. However utilizing SR-IOV will increase the performance because of direct access and can address scalability problem, but still it needs some considerations. The current major drawback is that live-migration is not possible. This is a great concern since it is totally against portability characteristics of VMs in a cloud environment. Furthermore since for implementing SR-IOV technique some device replacements are needed, the efficiency of utilizing this technique should be measured. Also achieved performance and its abilities should be analysed.

1.1 Problem Statement

According to what explained about efficiency and performance considerations in previous section, the considered problem is : To provide improved network performance and dynamic reconfiguration of underlying Infrastructures, for OpenStack IaaS cloud providers using SR-IOV capable Ethernet cards. This must be done in a transparent way to the end user.

- How much is the performance benefit of SR-IOV and pass-through comparing to emulated or paravirtualized network cards?
- How SR-IOV affects the matter of efficiency and Green computing in data centers?
- How to enable OpenStack to use SR-IOV for Virtual Machines automatically?
- How much is the network downtime of a Virtual Machine with a SR-IOV device attached when live-migrating?
- How to enable transparent-to-the-user live-migration of a Virtual Machine with SR-IOV devices attached?

1.2 Thesis structure

The layout of this thesis is as follows:
The background chapter (2.1) comes after this introduction, where related works and literatures are collected. It aims at giving a brief overview of the different tools used throughout the project, as well as relevant technologies. The methodology chapter (3) next gives an explanation of objectives and methods of the study and describes the project plan. It is included by some important parameters and calculations related to the method. It is followed by the results and analysis chapters (4 and 5) where the actual results are displayed and analysed in detail. Then in the discussion chapter (6), an overall evaluation of what has been done, problems encountered and future work is discussed. And last comes the conclusion where the questions asked in the problem statement section (1.1) are answered clearly. Also an appendix chapter with all of the important scripts created, some configuration files related to the testbed and some graphs can be found at the very end of this document.

Chapter 2

Background

2.1 Cloud Computing

Nowadays the buzzword of "Cloud Computing" is heard everywhere. In many businesses, academic environments and research organizations there are some people who are working on this concept extensively. As it was mentioned in last chapter some surveys show a considerable tendency to exploit cloud computing in organizations. Cloud computing is known as the sixth paradigm of computing technology. Figure 2.1 retrieved from Voas and Zhang [77] shows six paradigms in computing.

A cloud is an integrated pool of physical and virtualized resources (such as processing, storage, networking, application, development platforms, etc.). These resources can dynamically leased and released by cloud users. There are many definitions and descriptions for cloud computing with different expressions, but Peter Mell and Tim Grancein (2009) in their paper "Effectively and Securely Using the Cloud Computing Paradigm"[52] , suggested the following definition for cloud computing:

> *"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable and reliable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal consumer management effort or service provider interaction."*

This definition was also published under "Recommendations of the National Institute of Standards and Technology of U.S. (NIST)" as "The NIST Definition of Cloud Computing" in 2009 [53]. From this definition and other descriptions in different sources[16][44][25], it can be retrieved

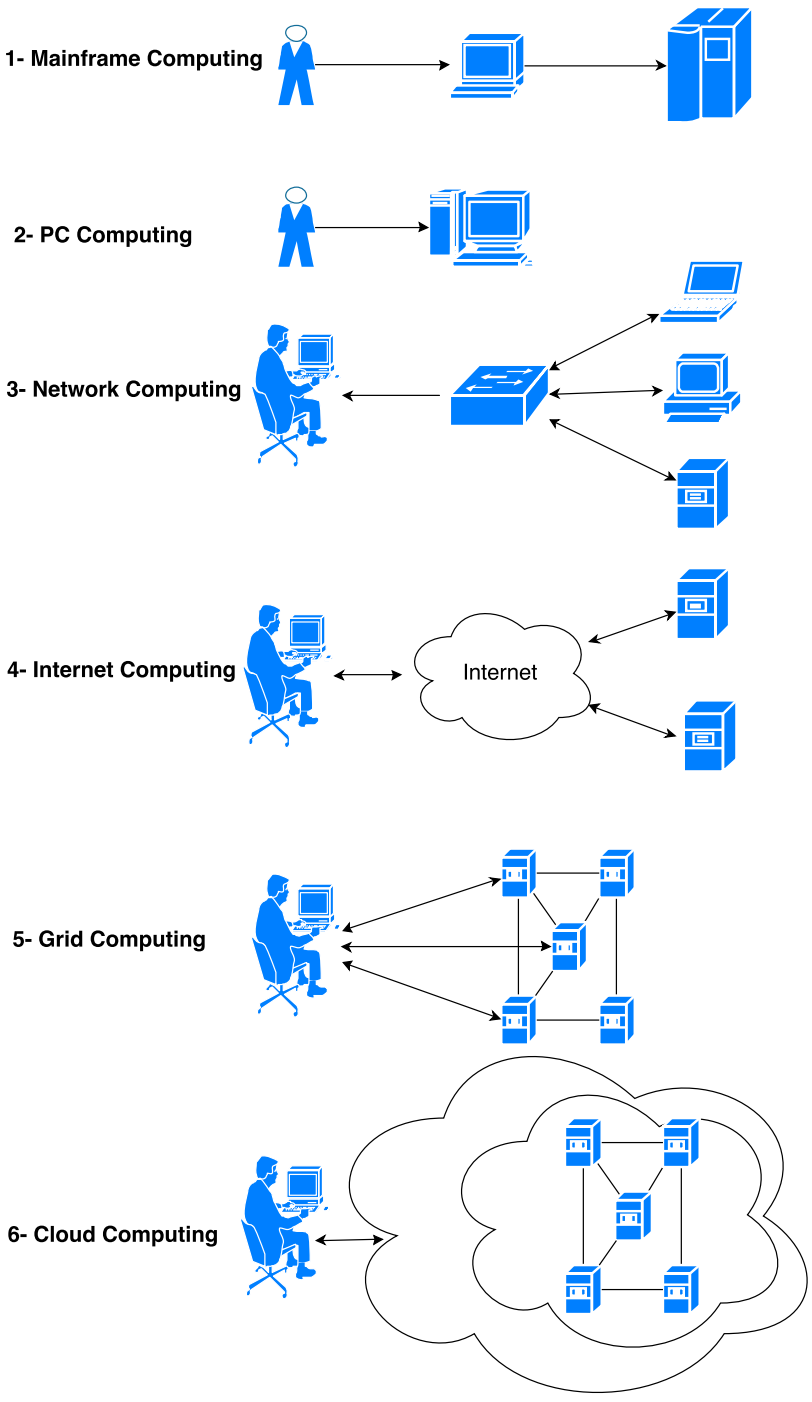


Figure 2.1: Computing paradigms shift

that by deploying clouds it is aimed to achieve scalable and highly available multiple computing services on demand through network (Internet).

2.1.1 Cloud Computing Features

The paradigm shift to the concept of cloud computing brought new computing characteristics. Cloud computing promises following features[16][82][44][3]

On demand services: All resources and services are provided based on user demand.

Pricing (Pay-as-you-go): Users do not need to do a big investment on infrastructure and technologies. They choose their needed services and just pay for time and/or amount of service which is delivered to them.

Scalability: Because of cloud resource pool and virtualized resources, all delivered services can be scaled up or down due to needs and/or request of cloud users.

High availability and reliability: By integration of big number of computing units, high performance networking and mass storages, clouds are able to provide a high quality of service (QoS). Moreover the reliability of services can be guaranteed by Inherent redundancy of cloud.

Efficiency: Clouds provide efficiency for both providers and users. There are many physical and virtualized resources which are dynamically allocated. Allocation of different resources (e.g. computing resource, infrastructures or even storages) is according to needs and based on dynamic allocation methods. This leads to use of resources in very efficient manner.

For example whenever a cloud user who has leased a virtual computer such as Amazon EC2, needs more resources such as more processing power, memory or storage, it is possible to reconfigure the virtual computer to add more resource. In the same way, then after a while it is possible to roll it back to main configuration. *Whenever* means on-demand and *reconfiguring* mean scalability. This user just pays normal fees for durations that the virtual computer is on and working, and amount of services

that are delivered. Payments just will increase for period of using more resources. This means pay-as-you-go! Always more resources are available for this user to request them and 24/7 working of the virtual computer is guaranteed. This implies Availability and Reliability of cloud. Leasing this virtual computer and using it just when it is needed instead of buying a power-PC for doing same task is more efficient to user

2.1.2 Cloud Architecture

From technical point of view clouds are divided to four layers but from service model point of view there are three layers [88][15][53][67][26]. In Figure 2.2 cloud layers are illustrated from both standpoints:

Hardware layer refers to physical resources of the cloud (data center equipments) such as physical servers, routers, cabling, etc.

Infrastructure layer refers to virtualized resources which are created on top physical resources by means of virtualization technologies.

Platform layer refers to operating systems and application development frameworks such as java, .Net, Python, etc. which all are installed on top of virtualized resources (infrastructure).

Application layer refers to all (cloud) applications which are placed on top of other layers to be delivered to users.

Services that are delivered by a cloud, are divided to three different service categories.

Infrastructure as a Service (IaaS) refers to on demand delivery of computing infrastructures like a service through the network or Internet. These services include virtual and physical machines (servers), storages and networking devices. The main benefit of IaaS for users is that they do not need to have a big initial investment on hardwares to run a project. This would be very suitable for short term or temporary projects developments. Also same as other services of cloud, pricing is according to usage amount and/or time. *Amazon Elastic Compute Cloud (EC2)* is the best example for IaaS. It delivers scalable computing capacity in the Amazon Web Services (AWS) cloud to users. Users can create and run *Virtual Machines Instances* with different configurations via web interface and Install and run their

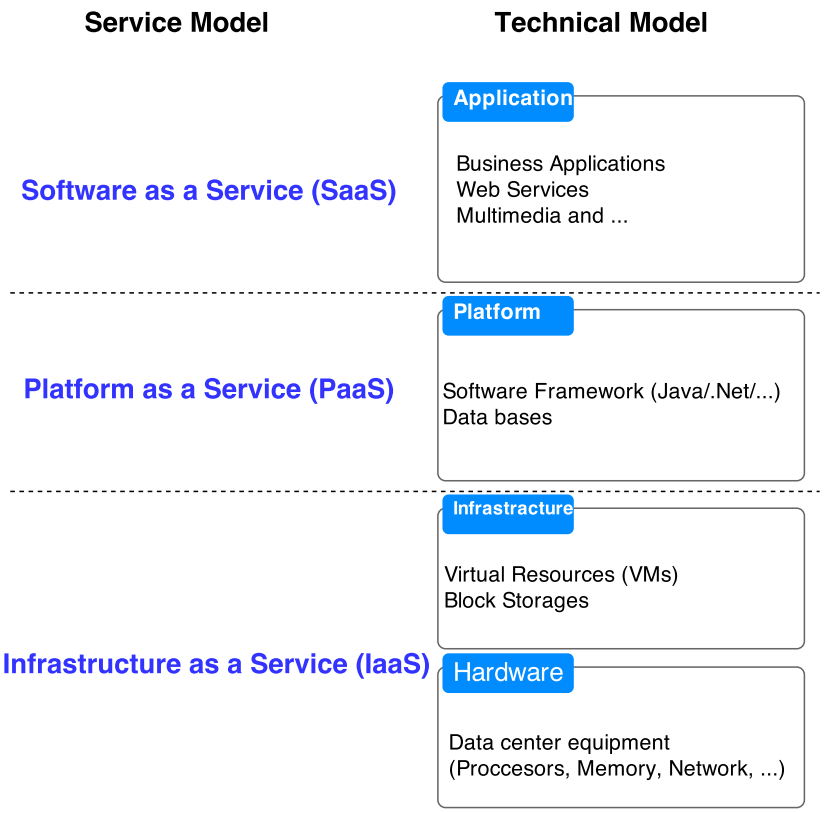


Figure 2.2: Cloud Computing architecture - Technical and service model view

own applications on it. Due to its fundamental role, IaaS is the most important part of a cloud. Moreover it is the most popular service of cloud especially for professional users/customers. Therefor the focus of this thesis falls under this category.

Platform as a Service (PaaS) refers to providing development environment like a service. PaaS is a virtualized platform that includes operating systems and specific applications (frameworks). *Google App Engine* [1], *force.com* and *Microsoft Azure* [2] are examples of this service layer which are tended by developers. Instead of buying one or some expensive licenses to develop a code, developers can rent services from PaaS providers to run, test and store code, applications and programs.

Software as a Service (SaaS) refers to on demand delivering of applications. This layer of service is the most visible layer to end users. The aim is to provide any application through web interface. Some of Google chrome apps [3], Google docs and even Gmail and facebook are examples of this layer of service. To date many different applications are delivered via Internet that all of them at the point of user view are platform independent. They can cover most of users needs by just having a web browser and an Internet connection.

2.1.3 Cloud Deployment Models

From deployment point of view there are three types of cloud.[53][26][91][16]

- Public Cloud
- Private Cloud
- Hybrid Cloud

What makes these types different is the way of provisioning cloud resources (computing resources).

Public cloud (also known as external cloud) is a type of cloud that all resources are dynamically provisioned off-site by a third party. Public

[1] https://developers.google.com/appengine/

[2] https://www.windowsazure.com/en-us/

[3] Those applications such as Box storage, sky Drive and some games which do not use local resources

cloud providers may offer different services from infrastructure to application via internet. Some of those providers deploy their cloud on top of services which are delivered by other providers. Most famous public cloud providers are Amazon, Google, VMware, Rackspace and Microsoft. Some providers like Amazon and Google offer different type of cloud services in different layers and some of them like Rackspace offer specific service . The main advantages of using public clouds is reduction of IT costs and risk transfer. By utilizing cloud infrastructure services (IaaS) instead of local data centers or other public cloud services, a big part of initial investments eliminates. The other thing is maintenance costs and issues which substantially slakes. This also leads to transferring risks from local businesses to cloud providers. But still there are some concerns about using public cloud services like security issues and connectivity problems.

Private cloud (also known as internal cloud) is called to cloud computing in an internal network or a cloud which is owned or leased by a specific organization. Usually this type of cloud is deployed on top of datacenter of organization for that specific business purpose. There are some different platforms (softwares) to deploy a cloud such as OpenStack, Open nebula, Eucalyptus Cloud Platform, VMware cloud, etc. Since there are some security considerations about public clouds services as well as the matter of cost-benefit, many businesses have deployed or are deploying their own cloud. Also many universities have been or are deploying their own private clouds for educational purposes.

Hybrid cloud refers to a combination of two other types of clouds. In Order to address some limitations of private clouds and some security threats of public clouds , a combination of them is used by some organizations. Hybrid clouds can offer better flexibility than private and public clouds but this combination may lead to more complexity.

The tools and technique used in this thesis, can provide for all mentioned types of cloud computing. It can address a common issue and offer better performance for all the three types.

2.2 OpenStack

OpenStack is a collection of open source components (Software projects) to build public or private clouds [55]. This collection of softwares also known as IaaS framework[41]. OpenStack is one of the more recent projects in cloud computing which has attracted very big attention to itself. As mentioned in motivation part, now more than 200 companies have joined the project and are in collaboration to develop it.

OpenStack IaaS framework consists of three core software projects, OpenStack Compute (known as Nova), OpenStack Object Storage (known as Swift), and OpenStack Image Service (known as Glance). Also there are other projects such as OpenStack identity service (known as Keystone), OpenStack Block Storage (known as Cinder), OpenStack Network (known as Neutron in the most recent release) and OpenStack Dashboard which is the web interface of cloud (known as Horizon). The software package is released on a six-month cycle that the first release was at October 21, 2010 under the name of Austin and the most recent one is released at February 13, 2014 under the name of Havana. Table 2.1 shows different releases of OpenStack until time of writing this thesis.

Table 2.1: OpenStack release history

Series	Status	Last Release	Date
Havana	Current stable release, security-supported	2013.2.2	Feb 13, 2014.
Grizzly	Security-supported	2013.1.4	Oct 17, 2013.
Folsom	EOL	2012.2.4	Apr 11, 2013.
Essex	EOL	2012.1.3	Oct 12, 2012.
Diabo	EOL	2011.3.1	Jan 19, 2012.
Cactus	Deprecated	2011.2	Apr 15, 2011.
Bexar	Deprecated	2011.1	Feb 3, 2011.
Austin	Deprecated	2010.1	Oct 21, 2010.

2.2.1 Components

Nova is the OpenStack compute service that contains subcomponents (nova-api, nova-compute, nova-scheduler). The Compute service is the main part of the IaaS and is the controller of the cloud. It is aimed to host and manage different cloud computing systems hence Nova interacts to other components like Keystone for authentication or Horizon (Dash-

board) for user interface. Nova conceptually is similar to Amazon EC2 and can leverage multiple hypervisors like Xen, KVM, etc. to build virtual instances and manage them. Nova also provides a basic networking service for virtual instances that is called Nova network. This simple flat networking has some limitations and can not support complicated networking needs. Therefore OpenStack has a specific component for advanced networking

Glance is the OpenStack image service that provides a repository for virtual disk images. This service enables cloud users to register new virtual disk images, queries for information on publicly available disk images, and the use of Glance's client library for streaming virtual disk images (Register, Discover and retrieve disk images).

Swift is the OpenStack object storage service which offers a highly scalable redundant storage system. By means of Swift clients of cloud are able to store and retrieve large amount of unstructured data with a simple API. Objects and files are stored to multiple disk drives spread throughout multiple servers in the data center.

Keystone is OpenStack identity service which provides policy and authentication service for other services in the cloud. It means that all other services relies on it for authentication and authorization of all API requests. Keystone provides the following concepts:

- **Identity:** provides authentication credential validation and data about Users, Tenants and Roles and other associated metadata.
- **Token:** validates and manages Tokens used for authenticating requests once a user/tenant's credentials have already been verified.
- **Catalog:** provides an endpoint registry to be used for endpoint discovery.
- **Policy:** provides a rule-based authorization engine.

Cinder is block storage service of OpenStack. It provides an infrastructure for managing volumes that offer persistent block storage to guest VMs in OpenStack. Before the Folsom release this service was originally a Nova

component called nova-volume, but due to complexity of nova it has become an independent project .

Horizon is a web application runs on apache that is the Dashboard of OpenStack cloud. It provides a modular web based user interface to other OpenStack services such as Nova, Swift, Keystone, etc. Horizon enables users to perform most cloud operations (e.g. launching an instance, managing networks and setting access controls) in an easy and graphical way. Also it helps users to monitor their resources and reading logs instantly.

Neutron Also know as Quantum [4] in older OpenStack releases, is the most recent release (at the time of writing this thesis) for networking component of OpenStack. Networking component is aimed to provide advanced networking service for IaaS elements. This component that can be called Networking as a Service, serves virtual networking devices within the cloud (e.g. vNICs and Virtual switches). In the other word neutron is responsible for defining virtual networks and connecting instances (created by Nova) to the virtual (software defined) networking infrastructure. There are three object abstraction in Neutron:

- **Network**
- **Subnet**
- **Router**

All of them acting exactly like their physical peer. Each configuration of Neutron may have one External network which represent a part of real external network and as many as required internal networks. Internal networks (also known as tenant networks) are assigned to tenants and may have some different subnets. VMs can not connect directly to external network but they can access it by traffic routing. Routers also are assigned to tenants and route traffics between internal and external network or between subnets. Figure 2.3 illustrates an example configuration of networking in OpenStack and figure2.4 illustrates an overall view of OpenStack cloud. Figure 2.3 that consists of VMs, virtual Networks, sub nets and virtual and physical routers, conceptually shows how networks are divided to different sub nets and how VMs are assigned to sub nets. It also shows how VMs in different sub nets might access VMs in other sub

[4] https://answers.launchpad.net/launchpad/+question/231396
https://lists.launchpad.net/OpenStack/msg24539.html

nets via routers and what is the role of virtual external network. Figure2.4 shows the relationships between different physical components of cloud as well as role and structure of different physical networks. Moreover it shows how virtual networks *seem to be* independent from physical networks and how they provide connectivity of VMs.

The main plug-in of Neutron is "***Open vSwitch***" which provides bridges and ports on different nodes. Ports on bridges are acting like physical switch interfaces and all instances are connected to ports. Bridges seems like an interface of the physical node and can be assigned by IPs of real networks of data center (e.g. external network). Neutron offers users two different "***GRE Tunnels***" and " ***vLANs***" configuration.
However GRE tunneling is more flexible and easy to configure but the encryption used by GRE may affect on performance. In GRE technique IP packet is wrapped to create new packet that has routing information. All packets are sent to network node of cloud (Neutron) and when wrapped packet reaches there, it is unwrapped, and the underlying packet is routed.Here tunnels are ports and bridges with a same name on different nodes appears as one bridge.
The Linux Bridge vLAN requires more complicated configuration. It modifies Ethernet header to add a 4-byte vLan tag and Open vSwitch interpret it. Packets tagged for specific VLAN are only shared with other devices belonged that VLAN.

2.3 Virtualization

As it mentioned already, virtualization is base of cloud computing. Some experts believe the concept of virtualization (as an idea of using a computer system to emulate another computer system) goes back to Mainframes era and it has a history as old as computing history [29] [79]. Basicly it was a method to divide mainframes logically to run multiple applications simultaneously. Over time due to hardware development and advances, especially processors, this concept has matured. It can be said that virtualization has been one of the key technologies to impact computing and nowadays significantly undertakes a critical role in world of computing.

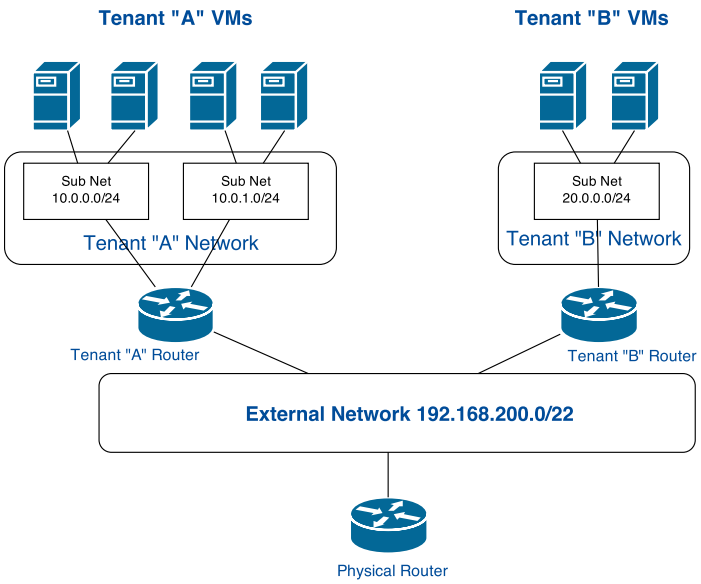


Figure 2.3: Sample configuration of networking by Neutron in OpenStack

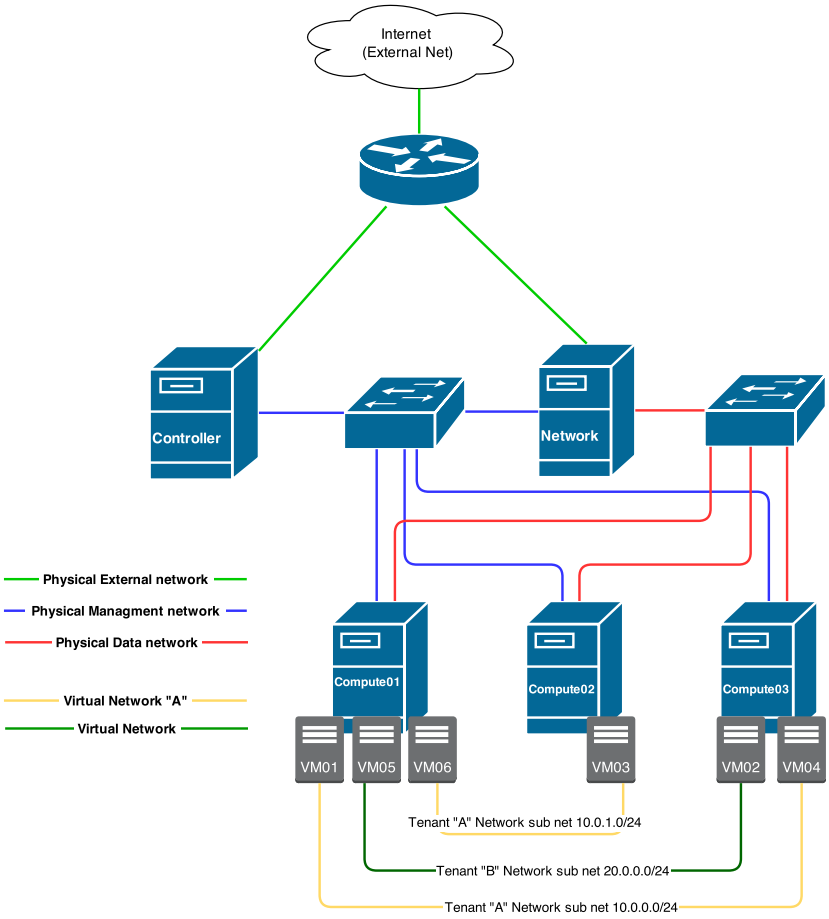


Figure 2.4: An overall view of sample OpenStack cloud

2.3.1 Definition

Singh, Amit (2004) presented this definition :

> *"virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine"* [69]

This definition is used in different publications[40][2][63][35][17].

In computing, Virtualization is process of emulating hardware or software resources by providing an abstract logical view of them. This emulation makes an isolated abstraction from the operating system on physical machine and underlying physical configuration. In another world virtualization is technology of creating virtual instances of objects, as an abstraction layer or environment, between hardware components and the end user . These instances that also known as Virtual Machine (VM) are more shareable, portable, manageable and flexible than peer physical objects.[18][70][79][39][29]. Common types of virtualization[66][17] can be classified as :

- **Infrastructure virtualization** (e.g. Network Virtualization and Storage Virtualization)
- **System Virtualization**(e.g. Server Virtualization and Desktop Virtualization)
- **Software Virtualization**(e.g. Application Virtualization and High-level language Virtualization)

2.3.2 Server Virtualization

Server virtualization[49][14] is the most common type of virtualization so that usually when someone talks about virtualization, means server virtualization. Simply it means partitioning a server by creating virtual instances of an entire server. It aims to hide the physical nature of server resources, such as the number and identity of processors, Memories, storages and other resources from the software running on them. Server virtualization has many benefits that the main is improving efficiency. By

direct utilization of a server the rate of utilization is just between 5% to 15% [78][49] of its abilities. Furthermore all applications has one-to-one relation with operating system and operating system has the same relation to hardwares. Server virtualization broke this monopoly and improves efficiency by abillity of having different operating systems and sharing resources through emulation and dynamic allocation.

There are two different techniques of server virtualization, *Type 1 (Native)* and *Type 2 (Hosted)* [9][32][51][45]. The platform of virtualizing server which lets virtual servers to be created and run on top of it, is called Hypervisor or Virtual Machine Monitor (VMM). Type 1 hypervisors are placed and run directly on top of hardware (physical server) and gets the control of resources. So they need to have some characteristics of an operation system through a kernel. Type 2 hypervisors are run as an application on top of host operating system of server. Type 2 heavily is depended on the operating system and its functionality so that if operating system experience a fault or crash it will effect on hypervisor and VMs. Figure 2.5 illustrates conceptual views of server virtualization.
Most famous hypervisors of each type are as follow:
Type 1 : Xen, KVM, VMware Esxi, MS Hyper-V
Type 2 : Oracle VirtualBox, MS Virtual PC, VMware Workstation

OpenStack cloud computing platform supports different hypervisors such as Xen, KVM, QEMU, Hyper-V, Esxi, etc. But the most compatible one is KVM and is configured as the default hypervisor for Compute[57][58]. For this thesis work OpenStack is chosen to provide IaaS testbed and KVM will provide virtualization platform in this cloud environment

2.4 I/O virtualization

One of the core applications of virtualization technology is to virtualize I/O devices. Inherited from general definition, I/O virtualization is the abstraction of upper layer protocols from physical I/O objects. With all virtualization technologies, the hypervisor must provide the guest operating systems with all devices that the it require to successfully run. In the other word to decouple VMs (logical) I/O devices from its physical implementation in the physical host server. This involves managing the

routing of I/O requests between virtual devices and the physical device. The aim is to provide guest virtual machines access to hardware devices to perform I/O. But any I/O virtualization solution should provide the same isolation that was found when the environment was running on a separate physical machine. There are multiple achievements from virtualizing I/O[65][80]:

- Multiplexing
- Isolation
- Portability
- Interposition

It can be said that the many advantages of virtualizing servers are caused by utilizing virtualized I/Os. The virtual I/O enables the hypervisor to share devices between multiple guest operating systems. This multiplexing virtual I/O devices onto physical I/Os leads to utilization of them at higher scale and achieve better hardware efficiency. Since the guest operating system is presented by virtual I/O it will be isolated from any changes and heterogeneity of underlying hardware. Furthermore while VMs are utilizing virtual I/Os they are so flexible to suspend and resume. Portability of VMs that results ability of Live Migration or is indebted to virtual I/Os. Despite all these positive characteristics, the imposed processing load of different I/O virtualizing is the main concerns in utilizing virtualization technique.[68][89][1]
There are three different approaches to I/O virtualization[11][39][32]:

- Full virtualization (Software Emulation)
- Paravirtualization
- Direct Access I/O (Pass-through)

The difference between these three approaches is mode of operation and interference of hypervisor.

2.4.1 Full virtualization(Software Emulation)

This technique aims to provide a full abstraction layer of the underlying physical devices and create a complete virtualized environment. Therefore this approach also known as Emulation or Software emulation. The

emulation layer interposes itself between the driver running in the guest OS and the underlying hardware. In this case the guest OS is totally unmodified that means it is not aware about virtualized environment and does not require any modification to work in this configuration. This is the only technique in virtualization which does not require any assist of hardware or operating system to virtualize sensitive and privileged instructions. Emulation layer can parse the I/O instruction of guest OS and translate guest addresses into host physical addresses. All guest OS instructions are translated *on the fly* by the hypervisor.
Advantages of full virtualization are easy setup and complete decoupling of virtual environment from physical devices. This leads to isolation and security for virtual machines, and also easier migration and portability. But this technique impose a considerable processing load because of device emulation by hypervisor. On the other hand usually the hardware device that is emulated is an older, generic device that supports various drivers across various operating systems. But It provides the ability for guest operating systems to run and use emulated devices with no special drivers and with no modification to the operating system.

2.4.2 Paravirtualization

Unlike full virtualization that whole system is emulated, paravirtualization provides an abstraction of each underlying device which is similar but not identical. It means that hypervisor disclose a modified version of the physical device interface to the guest VM. In this case the guest OS not only is aware that it is being virtualized and running on top of hypervisor but also it is modified and includes some custom device drivers[76]. In modified OSs the drivers of devices are replaced with calls to the hypervisor interface known as PV-drivers. This technique was introduced by Xen[5]and adopted by microsoft and VMWare in 2006 [5].
The aim in paravirtualization is to reduce the processing load of device virtualization by hypervisor. This is achieved by eliminating low-level emulation of devices and collaboration between guest OS and hypervisor. But still this approach has some disadvantage among which are Complex driver architecture and guest OS modifying issues.
In some solutions, such as Xen, the entire guest operating system is paravirtualized for an efficient, cooperative relationship with the hypervisor. In other solutions, such as VMware and KVM, only the device drivers are

[5]History of xen : http://www.xenproject.org/about/history.html

para-virtualized. In KVM, Virtio is chosen to be the main platform for I/O virtualizing (Paravirtualization). It provides an efficient abstraction for hypervisor and a common set of I/O virtualization drivers. virtio is an abstraction for a set of common emulated devices in a paravirtualized hypervisor. Its design allows the KVM to export a common set of emulated devices and makes them available through a common application programming.

2.4.3 Direct access I/O (Pass-through)

As the name of this technique shows it is totally different to both previous approaches. Direct access I/O or Device pass-through [6]means that the guest OS has direct access to physical device. In this case the hypervisor has no interference in I/O operations of guest OS and lets it pass through to the device.[89]. It means that the device can be assigned directly to VM and the guest OS drivers can communicate with device hardware directly without relying on any driver capabilities from the hypervisor or host OS. This achieves by means of I/O Memory Management Unit (IOMMU), that translates the I/O device DMA addresses to the proper physical machine addresses

The aim in this technique is to eliminate significant performance overhead of full or paravirtualization which is caused by hypervisor or host OS interference. Also by this direct access the performance of I/O operations increases. The performance of these operations are effective to total performance of VM so achieving a near native performance is a very positive point. But there are very significant disadvantages for Pass-through which are Non-portability and device sharing issue[19]. Since a VM is directly connected to the device and device is exclusively assigned to the VM neither the VM can be migrated nor the device can be shared with other VMs. These are against the scalability and portability aims of virtualization.

Figure 2.6 illustrates all three approaches together in a comparative way. It shows in each of technique how a VM can access the underlying device and how a hypervisor (and host OS) interferes in this process.

[6]In intel documents also known as direct assignment: http://www.intel.com/content/dam/doc/application-note/pci-sig-sr-iov-primer-sr-iov-technology-paper.pdf

2.5 KVM

Kernel-based Virtual Machine known as KVM[46] is one of the most recent type 1 hypervisors[10]. KMV has a significant difference with other type1 hypervisors that makes it unique. Normal native hypervisors are installed and run on top of hardwares without or independently from an operating system. A portion of them act as an operating system to access and control the machine resources that makes them big and complex. But KVM developers focused on other aspects of virtualization instead of developing a big part of an OS. Actually KVM is a kernel module for Linux operation system[46][39][90] that turn the operating system kernel to a hypervisor. This unique method of mixing hypervisors abilities with a host linux kernel leads to higher performance and simplexity[33][7]. On the other hand this technique always benefits from developments and advances of linux since there are ongoing works on linux kernel[7]. It should be mentioned that KVM solution is designed for virtualization on x86 hardwares[10] that contain virtualization extensions (e.g Intel VT or AMD-V). It means KVM is suitable to utilize only on machines that their processor supports hardware virtualization.

2.5.1 How does KVM work?

As it mentioned, KVM is a loadable kernel module for linux so can be installed from most of repositories[33][47]. An installation of KVM consists kvm.ko module, aims providing core virtualization infrastructure and a specific processor module, *kvm-intel.ko* or *kvm-amd.ko*. In order to create virtual machines, KVM uses modified version of QEMU hardware virtualizer by maintaining a fork of it called *qemu-kvm*.

QEMU is a generic and open source machine emulator and virtualizer[84]. In a KVM installation it is a user-space component for emulating machine devices that provides an emulated BIOS, PCI bus, USB bus and a standard set of devices such as IDE and SCSI disk controllers, network cards, etc. QEMU afford near native performances by executing the guest code directly on the host CPU. While utilizing KVM, created virtual instances are as regular Linux processes that are scheduled by operating system scheduler (Linux scheduler). These processes have specific execution mode, added by KVM to OS, called *guest mode*. Normal execution modes

of processes in a Linux operating system are User mode and Kernel mode which user mode is the default mode for applications. An application change into kernel mode only if it require a service from kernel, such as an I/O service. The added guest mode also has both user and kernel types inside its space. A process with guest execution mode is a process that is run from inside of a virtual machine. Figure 2.7 shows a conceptual view of KVM virtualization architecture.

By installing KVM on a Linux OS a hardware file */dev/kvm* is created that acts as interpreter between actual hardware and hypervisor. This file enables QEMU to send requests to KVM to execute hypervisor functions. KVM device node (/dev/kvm) provides following operations:

- Creating new virtual machines.
- Allocating memory to a virtual machine.
- Reading and writing virtual cpu registers
- Injecting an interrupt into a virtual cpu.
- Running a virtual cpu

The generic KVM command interface is provided by virsh (Virtualization Shell). By means of this shell it is possible to manage hypervisor and VMs directly from Host OS terminal. Virsh is built on top of libvirt library. This library is a Linux API over the virtualization capabilities of Linux that supports different hypervisors, such as Xen and KVM, QEMU and some virtualization tool for other operating systems.
In this work, KVM is chosen to utilize as VMM of OpenStack cloud testbed of project.

2.6 SR-IOV

As it mentioned already there are different I/O virtualization techniques (e.g. full and paravirtualization) to provide I/O devices for a VM. Each technique has some advantage as disadvantages among which processing overhead or sharing limitation. In a cloud environment performance, scalability and portability of VMs and sharing of resources are very important. Any technique and solution should be able to cover all of them in an acceptable way. Since the number of VMs per server can significantly

increase due to advances in processors, memory allocation and storages, I/Os (e.g. Network card interface) are going to be bottlenecks. Therefor Issue of virtualizing and sharing I/Os have been considered by researchers and producers. They believe that new devices should cooperate with hypervisor in virtualizing and be natively shareable.
In this regard PCI-SIG [60] introduced a new technique for I/O virtualization known as Single Root I/O Virtualization (and sharing [7]) i.e. SR-IOV[19][30]. Specification of SR-IOV defines a standardized mechanism to create natively shared devices that also handle a part of virtualization [61].

> *SR-IOV proposes a set of hardware enhancements for the PCIe device, which aims to remove major VMM intervention for performance data movement, such as the packet classification and address translation. SR-IOV inherits Direct I/O technology through using IOMMU to offload memory protection and address translation.*[19]

SR-IOV introduces two new function types *Physical Functions (PFs)* and *Virtual Functions (VFs)*
PFs: These are full PCIe functions that include the SR-IOV Extended Capability. The capability is used to configure and manage the SR-IOV functionality.
VFs: These are *lightweight* PCIe functions that contain the resources necessary for data movement but have a carefully minimized set of configuration resources.

A SR-IOV-capable devices provide configurable numbers of independent VFs, each with its own PCI Configuration space. So that each VF has its own requestor ID and resources. This allows a VF to be assigned directly to a VM and guest access the physical resource without intervention. VF specific requestor ID allows the hardware IOMMU to convert guest physical addresses to host physical addresses[89]

Since the SR-IOV inherits its functionality from Direct Access model, still the portability is a big concern. VMs that are directly connected to NIC have should be disconnected from network while migration. In this thesis this issue will be studied, a solution to address this problem will be implemented and results will be analysed.

[7] Used in Intel documents

2.7 Related works

Some research and studies have done both aimed at introducing technique or solution to live migration with direct access I/Oor utilizing SR-IOV to provide high performance network. There are some papers found addressing the same topic or related topics.

2.7.1 Live Migration with Direct access Device

Live Migration with Pass-through Device for Linux VM[89]
In this study Edwin Zhai, Gregory D Cummings and Yaozu Don introduced pass-through device and SR-IOV techniques and showed that direct access to physical device by VM leads to achieve close to native performance, but it is against live migration aim of virtualization. They introduced virtual ACPI hotplug device model that allows VM to hot remove the pass-through device before relocation and hot add another one after relocation. They enable continuous network connectivity for directly assigned NIC devices by integrating the Linux bonding driver into the relocation process.

Live Migration of Direct-Access Devices[37]
In this study Asim Kadav and Michael M Swift introduced shadow driver and described using this drivers to migrate the state of direct-access I/O devices within a virtual machine. They said however they implement shadow driver migration for Linux network drivers running over Xen, but it can be readily ported to other devices, operating systems, and hypervisors.

2.7.2 Improve virtualization performance with SR-IOV

High Performance Network Virtualization with SR-IOV[19]
In this study Yaozu Dong et al. introduced SR-IOV then designed, implemented, and tuned a virtualization architecture for an SR-IOV-capable network device, which supports reusability of PF and VF drivers across different hypervisors. They showed that the most time consuming tasks in interrupt handling are emulation of guest interrupt mask and unmask operation and End of Interrupt (EOI). In their implementation, they con-

ducted performance measurement to compare SR-IOV solution with others. It proved that SR-IOV provides a good solution for a secure and high performance I/O virtualization.

Improving Virtualization Performance and Scalability with Advanced Hardware Accelerations[21]
In this study Yaozu Dong et al. implemented and optimized the support of advanced hardware accelerations in the latest version of Xen, including Pause Loop Exit (PLE), Extended Page Table (EPT), and Single Root I/O Virtualization (SR-IOV). They showed that experimental results demonstrate very good performance and scalability on the multi-core and overcommitted system, for both micro-benchmark and a server consolidation benchmark. The results show an up to 77% improvement in the server consolidation benchmark (49% of which due to EPT and another 28% due to SR-IOV), and an up to 14% improvements in the micro-benchmarks due to PLE.

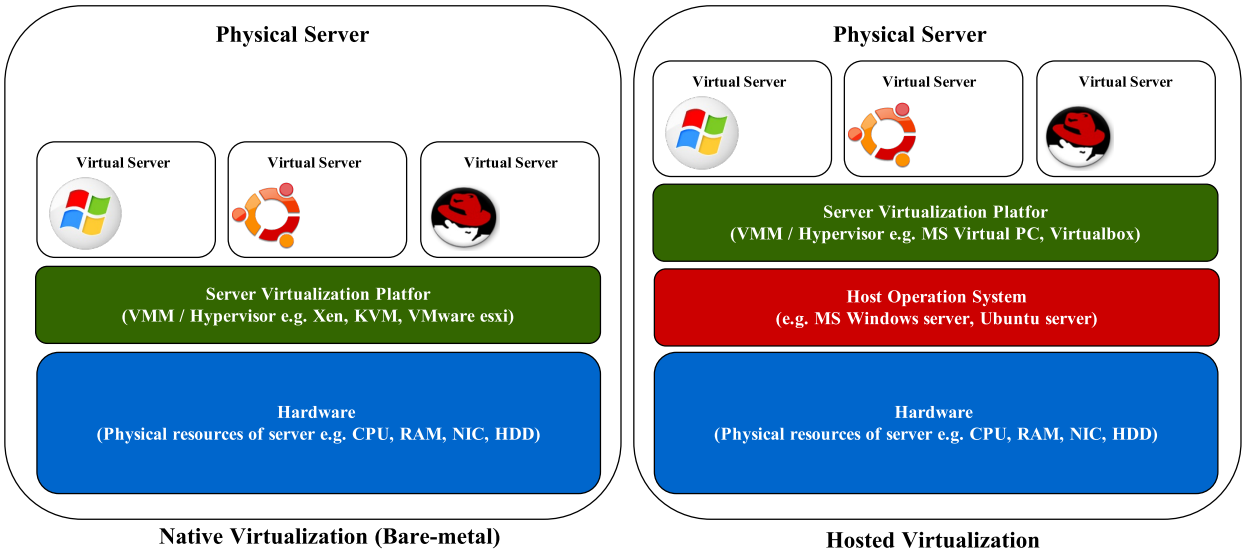


Figure 2.5: Conceptual view of two different server virtualization methods

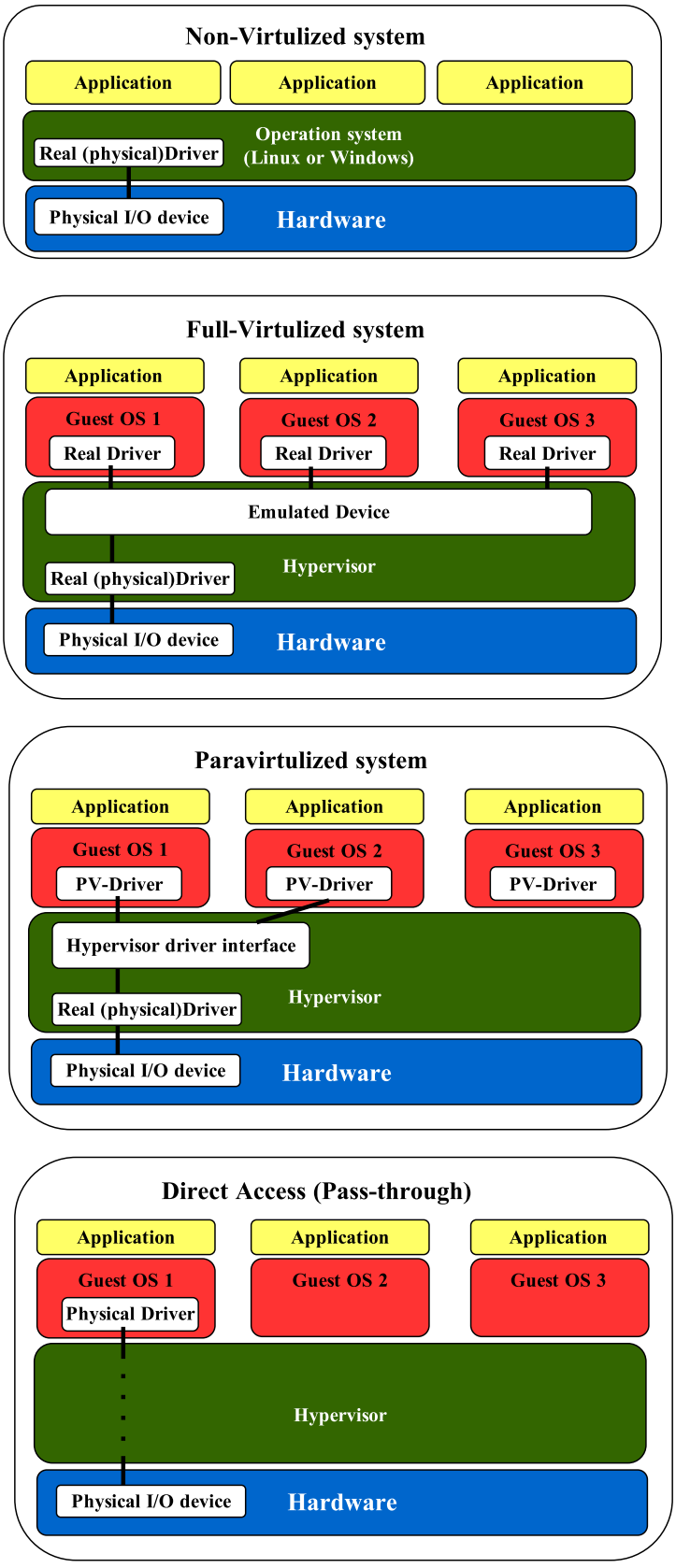


Figure 2.6: Conceptual view of Different approaches of I/O virtualization

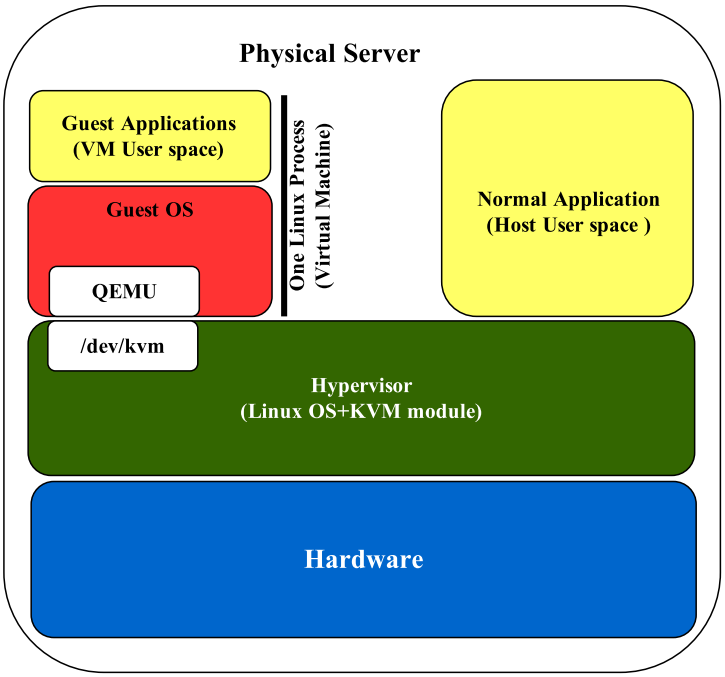


Figure 2.7: Conceptual view of KVM virtualizations - User-space and Guest space

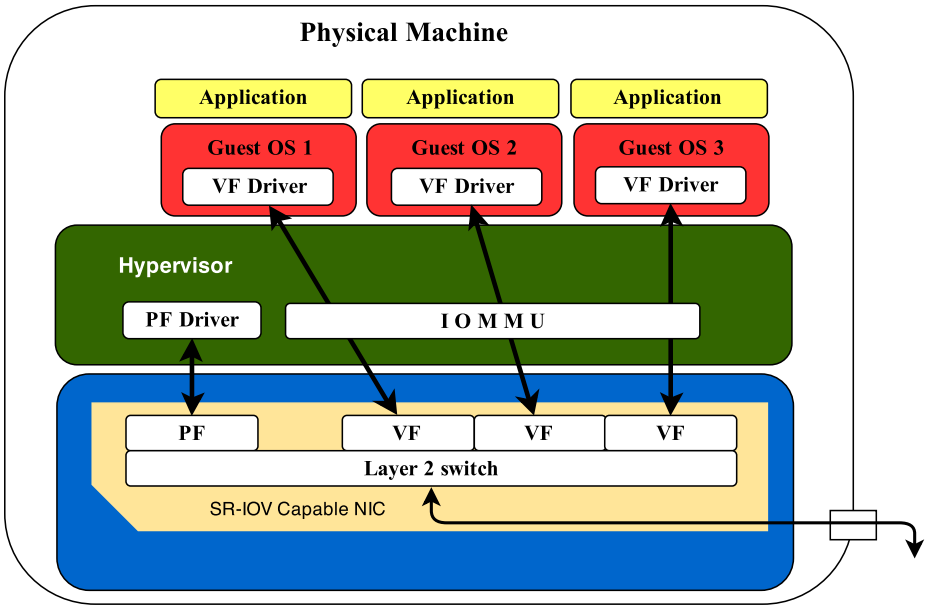


Figure 2.8: SR-IOV architecture - Assignment of PF and VFs

Part II

The project

Chapter 3

Methodology

The methodology chapter will explain the approach to the problem statement and addressing research questions including environment design, Hardwares and tools, planned workflow and the analytical procedures to achieve the final goal.

3.1 Objectives

Based on the problem statement of section 1.1, this study aims to address the issue of improving performance of networking in a cloud environment by utilizing a proposed method. In this solution the matters of efficiency, scalability and transparency are considered as much as possible.

3.1.1 Proposed method for migrating a SR-IOV attached VM

As it already stated in motivation chapter 1 and sections 2.4 and 2.6 in background chapter, SR-IOV technique is introduced to reduce the I/O virtualization overhead. It is designed to be scalable to some extend [1] and deliver high-performance networking. But the main concern about utilizing this technique is portability (or dynamic reconfiguration) which specifically affects the feature of Live-migration in a cloud environment.

This study tries to eliminate this umbarge by combining SR-IOV technique with a linux feature known as *linux ethernet bonding driver*[38] and

[1] Depending on NIC model the number of VFs is varied but it is limited

demonstrate the result. The Linux bonding driver provides aggregation of multiple network interfaces into a single logical interface known as *bond*. The method already was proposed by Edwin Zhai, Gregory D. Cummings, and Yaozu Dong (2008) [89] and Dong, Yaozu et al (2012) [20]. This project aims to implement the above idea in an openstack environment to improve the networking performance and enable the live migration of VMs while utilize it.

SR-IOV virtual function is attached to VM like a direct I/O and caused some constraints. Linux bonding driver can be configured to give the ability of having a reserved NIC as a slave. This slave should not be in use and does not interfere to networking, unless the primary NIC become unavailable. This combination can be used to prevent network loss while migration of VM with a direct access NIC. The SR-IOV VF is detached from VM in the origin to enable live migration without any error, and will be attached immediately at the destination from resources of destination host. During this short period the slave interface take care of connectivity. In this project the plan is to combine a SR-IOV interface with a Paravirtualized one using Linux bonding driver inside the VMs. Both interfaces should have been added to the VM while creating the VM. To implement this idea it is needed to create a same bridge interface in all hosts (Compute nodes). Same means to have same name and to be connected to a same network (be attached to network interfaces on same network). The Paravirtualized interface which is connected to this Linux bridge will be inactive all the time except the period of live migration that SR-IOV will be detached from VM. Figure 3.1 illustrates the the schema of the system and the connections after implementing this method.

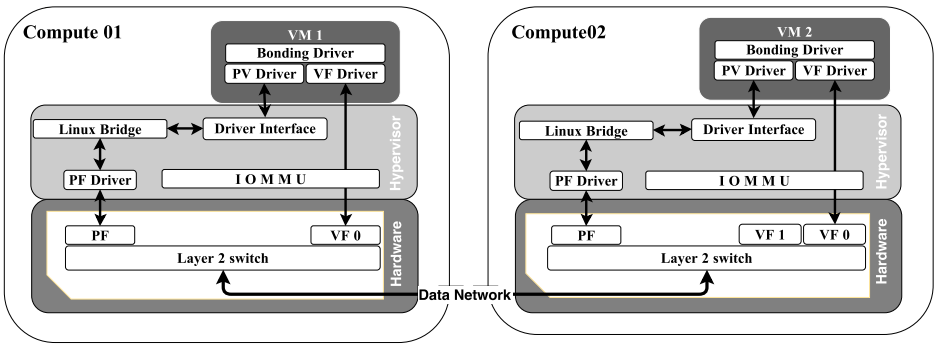


Figure 3.1: The schema of the environment by implementing proposed method of migrating SR-IOV attached VM

In above system by detaching SR-IOV VF from each of VMs, the connectivity will be established through Paravirtualized device and Linux bridge. Details of implementing this method in OpenStack will be explained in section Dynamic reconfiguration in SR-IOV enabled Openstack 4.3 in following chapter 4.

3.1.2 Investigation on different methods

Before demonstrating the implementation and utilization of above method an investigation has been done to observe and analyze the efficiency of all available techniques. This helps to estimate the costs and achievements by implementing proposed method. Also it helps to find best option for the slave NIC which would have minimal reduction in performance while switching to it. In this investigation all three approaches to I/O virtualization (mentioned in section 2.4) plus SR-IOV were implemented and benchmarked.

3.2 Testbed

To approach to the problem statement, this study first of all needed to setup and configure a test bed. *OpenStack* cloud platform was chosen to be used as the testbed for all of the experiments. In order to enable live migration functionality, the cloud was setup with two compute nodes, which is the minimum requirement for such a demonstration.

3.2.1 Hardwares

Physical equipments were including four server machines. The following table 3.1 shows the technical informations of the equipments used as infrastructures of testbed :

Table 3.1: Physical Servers

Brand	CPU	Cores	Memory	NICs
Sun - sunfire X2270	Intel®Xeon E5504	1x4	6 GB	2x1 Gb
Sun - sunfire X2270	Intel®Xeon E5504	1x4	6 GB	2x1 Gb
HP - ProLiant-DL360	Intel®Xeon E5-2609	2x4	32 GB	4x1 Gb 2x10 Gb SR-IOV
HP - ProLiant-DL360	Intel®Xeon E5-2609	2x4	32 GB	4x1 Gb 2x10 Gb SR-IOV

Equipments also included a 1Gb HP Ethernet Switch used to provide connectivity between physical nodes. HP servers that were equipped by SR-IOV capable NICs were used as compute nodes and SUN servers were used as controller and network node.

3.2.2 Infrastructure Design

As it mentioned already the infrastructure was used to deploy an Openstack cloud. To follow a standard design of cloud environment, the cloud contained one controller node to control and manage nodes and services, one network node to provide networking service of the cloud and two compute nodes to take care of virtual machines. All of the nodes were connected to the *management network* but Network node and compute nodes also were connected to another network called *Data Network*. Since in this design the Network node needs to have 3 NICs, one external USB 1Gb NIC was added to one of the sun servers.

Figure 3.2 shows the role of each server and the networks. The actual installation and setup of this cloud will be presented in the "System Setup" section 4.1.

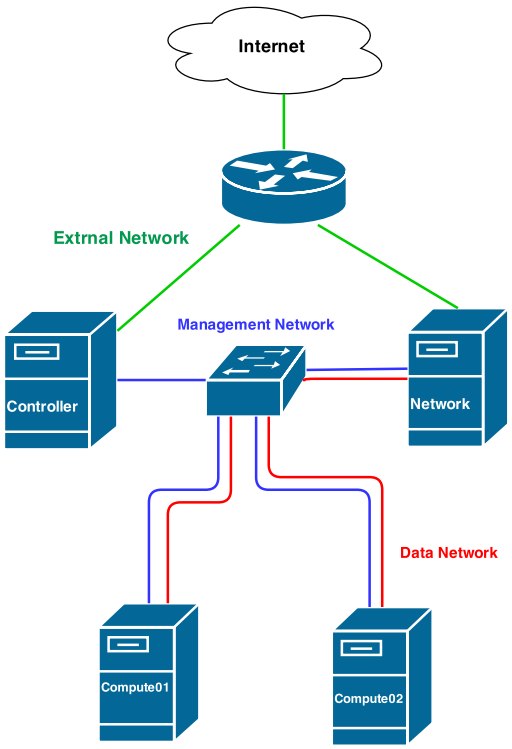


Figure 3.2: Overview of the Infrastructure Design

3.3 Experiments

To design experiments, it is necessary to recognize the affecting factors and key parameters which are going to be evaluated. The important factors identified for this experiment are briefly explained in the following section 3.3.1.

3.3.1 Experiment Factors

- **Bandwidth:** is a measurement of bit-rate of available or consumed data communication resources expressed in bits per second or multiples of it (bit/s, kbit/s, Mbit/s, Gbit/s, etc.)[74]. Usually the expected bandwidth might be different to nominal bandwidth since it is depended on other contributors in the network (connection) like cables, switches, other nodes, etc. Also sometimes the delivered bandwidth is different to expected bandwidth due to environmental effects such as noises, collision and intentional traffic decline by load balancers [31].

- **Downtime:** Refers to the period of time that the service is not entirely ready or available [12]. In this study the duration of disconnecting network and connecting it again at the destination while live migration is intended. This duration can be count by milliseconds. It might happen either intentionally like what happens to SR-IOV interface in the method of this study or unintentionally which might caused by a failure during live migration.

- **Network Performance:** Here in this study the delivered bandwidth of each configuration is intended. It is compared with the theoretical expected bandwidth according to the hardware specifications as well as delivered bandwidth by other configurations in a same condition. The actual bandwidth (which also known as throughput) of NIC can be calculated from amount of data that is continuously transferred during each test period. Also the configurability and stability of the network could be considered. The stability is measured by calculating the mean and standard deviation of average bandwidth of each single test. This is going to be explained in more detail in following section.

- **Overhead:** This term refers to processing overhead of different configurations. The extra processing load and memory usage which are imposed to the system (Compute node) by each of I/O virtualization techniques are intended. the overhead can be calculated by monitoring processing resources of system and memory during all experiments and comparing the results together.

- **Efficiency:** Considering delivered bandwidth ratio to expected bandwidth in a method and taking imposed overheads and energy cosumption to the account, would show the efficiency of the method.

- **Transparency:** The term refers to transparent procedures to the user level. The aim is to design and run procedures which applying reconfigurations without any interfer or with very few interference to the user space.

3.3.2 Experiment Design

According to problem statement considerations and based on the above factors the following experiments designed to investigate performance of each technique as well as behaviour of system while utilizing them.

1. **Single VM:** All four methods of I/O virtualization including *Emulation, Paravirtualization, PCI passthrough* and *SR-IOV* were separately configured on a same VM. The reason of using same VM is to have quite same environmental conditions during all test. Frequently the performance of network had been benchmarked by pushing as much as possible load to the NIC. Meanwhile for further investigation a bunch of system indicators (including CPU usage, system load, Memory usage and Power/Energy consumption) were recorded in all systems that were involved in tests. The aim is to measure and compare real throughput of each method and its impacts on system. These experiments introduced the method which ideally has the best performance.

2. **Multiple VMs:** Three methods of *Emulation, Paravirtualization,* and *SR-IOV* were separately configured on multiple VMs which were hosting in a same compute node. The same benchmark, as the single VM experiment, was done on each of VMs and same system indicators were recorded. The aim of this type of experiment is to investigate the scalability of methods and observe the impact of multiple usage of method on the system. These experiments helps to measure efficiency and stability of each configuration more precisely. It should be mentioned that the passthrough technique is not scalable since it occupies one physical interface and there is a limit number of NIC on physical machines. So this experiment can not be done by passthrough technique.

3. **Live Migration demonstration:** After comparing performance of each method, two experiments were done based on live migration. Since the ability of live migration while using Emulated and Paravirtualized I/O is already known and clear, so these type of experiments were done to show how a VM with SR-IOV attached interface should be migrated. The first experiment was doing live migration of a VM in different conditions to measure the downtime of network. Different conditions means to have different loads that lead to have different migration time. The second experiment is demonstrating live migration while using the proposed method of this study to eliminate downtime of ethernet SR-IOV and measure the performance penalty. This experiment aims to demonstrate the ability of live migration with SR-IOV.

3.3.3 Tools and Scripting Languages

In this study three types of tools were needed to do the project :

1. **Monitoring (Data gathering) tools:** This type of tools are needed to monitor system indicators. They are supposed to gather data from different files and record them in file for further analysis. Therefore two scripts (loads.pl and power.pl) were written in *perl* to be used in different machines. Details are explained in section 3.3.4 and the result chapter.

2. **Benchmarking tool:** A tool was needed to impose as much as possible traffic to the NICs. *Iperf*[28] is found a suitable tool for this purpose and has been used in this project. It is a tool to measure the maximum achievable bandwidth on IP networks which supports tuning of various parameters related to timing, protocols, and buffers. iperf push a heavy load to the network and reports the bandwidth, loss, and some other parameters. To benchmark the network, it needs to be run as a front end (client) and a backend (server) on different nodes and measures the bandwidth between this two nodes. A sample configuration and output of iperf is as bellow:

Sample iperf - Server side command and result

```
root@controller:~# iperf -p 12000 -s
------------------------------------------------------------
Server listening on TCP port 12000
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.10.1 port 12000 connected with 192.168.10.20 port 55116
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 5.1 sec  57.0 MBytes  93.9 Mbits/sec
root@controller:~#
```

Sample iperf - Client side command and result

```
root@network:~# iperf -p 12000 -c 192.168.10.1 -t 5 -i 1
------------------------------------------------------------
Client connecting to 192.168.10.1, TCP port 12000
TCP window size: 22.9 KByte (default)
------------------------------------------------------------
[  3] local 192.168.10.20 port 55116 connected with 192.168.10.1 port 12000
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  12.0 MBytes   101 Mbits/sec
[  3]  1.0- 2.0 sec  11.1 MBytes  93.3 Mbits/sec
[  3]  2.0- 3.0 sec  11.2 MBytes  94.4 Mbits/sec
[  3]  3.0- 4.0 sec  11.4 MBytes  95.4 Mbits/sec
[  3]  4.0- 5.0 sec  11.1 MBytes  93.3 Mbits/sec
[  3]  0.0- 5.0 sec  57.0 MBytes  95.2 Mbits/sec
root@network:~#
```

In above sample first the server side configured to listen on tcp port 12000 and then the client side configured to send traffic to the server (192.168.10.1:12000) for 5 seconds and report bandwidth in 1 second intervals.

However iperf measures the throughput of the network and report it but in order to have more accurate results the throughput was measured from output of monitoring tools.

3. **Automation tools (Utilities):** Some tools were needed to do bunch of tasks to automate experiments. Therefore couple of scripts were written using *shell scripting* to create VMs, Installing needed packages and configure them (i.e time synchronization, network configuration), copying tools to different machines, running experiments and gathering outputs.

   Also some tools were needed to automate the process of attaching or detaching SR-IOV NICs to/from VMs. At the time of writing this thesis using SR-IOV was not directly supported by openstack (Neutron). Mellanox had introduced a plugin [2] for its products to handel using of SR-IOV with neutron. Due to time limitations and other concerns (needed changes, reconfigurations and additional installation requirments as well and specific drivers.[3]) the decision was not testing usage of that plugin in this project. So five scripts were written using shell scripting to handle this matter and using SR-IOV VFs for cloud instances. These scripts collaborate with openstack nova to attach SR-IOV VFs to the VMs at the time of creating VM, detach it before migration and reattach it after migration at the destination. In this process it is needed to generate a unique MAC address for VFs, finding the first available PCI address of VFs (in the correct host), creating related xml file and attach it to the target instance. Needed information are taken from nova and actions are done using libvirt.

[2] https://wiki.openstack.org/wiki/Mellanox-Neutron-Havana-Redhat
[3] This will be explained in discussion section

3.3.4 Data Collection and Evaluation

While running comprative experiments, the *loads.pl* script had duty to gather data related to system loads and record them in a comma seperated output file. Data were frequently gathered from files under */proc* including memory , load, CPU usage and Network activities data.

The file */proc/stat* contains various pieces of information about kernel activities[4]. All the informations are aggregated since the system is booted. Multiple first lines contain amount of time (hundreds of a second) that different CPUs (Cores) have spent performing different kinds of tasks as well as being idle. The recorded informations by loads script are as bellow:

```
- user: normal processes executing in user mode
- nice: niced processes executing in user mode
- system: processes executing in kernel mode
- idle: twiddling thumbs
- iowait: waiting for I/O to complete
- irq: servicing interrupts
- softirq: servicing softirqs
- steal: involuntary wait
- guest: running a normal guest
- guest_nice: running a niced guest
```

A sample content of this file is like:

```
cpu  8634522 66570 1038388 42579703 241324 20 18343 0 0 0
cpu0 4297158 30687 595306 42111124 240535 20 17881 0 0 0
cpu1 4337363 35882 443082 468579 788 0 462 0 0 0
intr 105385792 66 51 0 0 0 0 0 1 3 0 0 52 0 463519 0 10197096 0 181086 0 1374662 29313
1201132 356678 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ctxt 293525297
btime 1397382840
processes 53743
procs_running 1
procs_blocked 0
softirq 65112485 16 30578308 95482 8829012 1606828 16 269105 14854146 39001 8840571
```

The very first line presents the sum of other following lines started by cpu#. The percentage of CPU usage is calculated from those information

[4]http://www.linuxhowtos.org/System/procstat.htm

by following formula[5]:

$$Usage\% = 100 \times \frac{(\mathrm{d}total) - (\mathrm{d}idle)}{(\mathrm{d}total)}$$

or

$$Usage\% = 100 \times \frac{(total\ time\ of\ cpu\ during\ the\ period) - (idle\ time\ of\ cpu\ during\ the\ period)}{(total\ time\ of\ cpu\ during\ the\ period)}$$

The *total time of cpu during the period* refers to sum of all the times which CPU either is doing different tasks or is idle. The idle time of cpu during the period refers to sum of the times which CPU is idle for any reason. According to following piece of Linux kernel code, the *guest* and *guest_nice* times already are added to *user* and *nice* time of cpu, so it is not needed to count them again while calculating CPU total time.[6]

```
static void account_guest_time(struct task_struct *p, cputime_t cputime,
                               cputime_t cputime_scaled)
{
        u64 *cpustat = kcpustat_this_cpu->cpustat;


        /* Add guest time to process. */
        p->utime += cputime;
        p->utimescaled += cputime_scaled;
        account_group_user_time(p, cputime);
        p->gtime += cputime;


        /* Add guest time to cpustat. */
        if (task_nice(p) > 0) {
                cpustat[CPUTIME_NICE] += (__force u64) cputime;
                cpustat[CPUTIME_GUEST_NICE] += (__force u64) cputime;
        } else {
                cpustat[CPUTIME_USER] += (__force u64) cputime;
                cpustat[CPUTIME_GUEST] += (__force u64) cputime;
        }
}
```

Also the following piece of Linux kernel code shows that *iowait* is not added to *idle* which should be calculated as idle time of CPU.

[5]http://www.design-reuse.com/articles/8289/how-to-calculate-cpu-utilization.html

[6]The code is retrieved from : http://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/kernel/sched/cputime.c?id=ec6931b281797b69e6cf109f9cc94d5a2bf994e0

```
void account_idle_time(cputime_t cputime)
{
        u64 *cpustat = kcpustat_this_cpu->cpustat;
        struct rq *rq = this_rq();

        if (atomic_read(&rq->nr_iowait) > 0)
                cpustat[CPUTIME_IOWAIT] += (__force u64) cputime;
        else
                cpustat[CPUTIME_IDLE] += (__force u64) cputime;
}
```

According to implementation of *htop* [7] and above matters, the final formula to calculate CPU usage percent during a period should be as bellow[8]:

$$PrevIdleTime = previdle + previowait$$

$$IdleTime = idle + iowait$$

$$PrevNonIdleTime = prevuser + prevnice + prevsystem + previrq + prevsoftirq + prevsteal$$

$$NonIdleTime = user + nice + system + irq + softirq + steal$$

$$PrevTotalTime = PrevIdleTime + PrevNonIdleTime$$

$$TotalTime = IdleTime + NonIdleTime$$

$$Usage\% = 100 \times \frac{(TotalTime - PrevTotalTime) - (IdleTime - PrevIdleTime)}{(TotalTime - PrevTotalTime)}$$

The other indicator to consider for calculating system overheads is memory usage. The file */proc/meminfo* reports a large amount of information about the Linux systems memory. Among those information the loads script record *MemTotal*, *MemFree*, *Buffers* and *Cached*. The content of these values are as below:

- **MemTotal:** Total usable RAM in kilobytes (i.e. physical memory minus a few reserved bytes and the kernel binary code)

- **MemFree:** The amount of physical RAM left unused by the system.

[7] https://github.com/hishamhm/htop/blob/master/ProcessList.c

[8] http://stackoverflow.com/questions/23367857/accurate-calculation-of-cpu-usage-given-in-percentage-in-linux/23376195#23376195

- **Buffers:** The amount of physical RAM used for file buffers.

- **Cached:** The amount of physical RAM used as cache memory. Memory in the pagecache (diskcache) minus SwapCache.

A sample content of this file is as like :

```
MemTotal:        3589204 kB
MemFree:          208888 kB
Buffers:           45536 kB
Cached:          1329460 kB
SwapCached:       151888 kB
Active:          2015588 kB
Inactive:        1142500 kB
Active(anon):    1491664 kB
Inactive(anon):   633688 kB
Active(file):     523924 kB
Inactive(file):   508812 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:       3657724 kB
SwapFree:        3337268 kB
Dirty:               172 kB
Writeback:             0 kB
AnonPages:       1634320 kB
Mapped:           172516 kB
Shmem:            342260 kB
Slab:             137668 kB
SReclaimable:      94836 kB
SUnreclaim:        42832 kB
KernelStack:        3040 kB
PageTables:        36924 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     5452324 kB
Committed_AS:    3514176 kB
VmallocTotal:   34359738367 kB
VmallocUsed:      347136 kB
VmallocChunk:   34359383676 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       64024 kB
DirectMap2M:     3596288 kB
```

The actual memory usage during a period or at any time can be calculated through following formula:

$$Used\ Memory = Total\ Memory - (Free\ Memory + Buffers + Cached)$$

System load average is the third indicator taken by the loads script. In UNIX computing, " *the system load is a measure of the amount of computational work that a computer system performs*"[9]. The script gather this information from file */proc/loadavg*. The first three fields in this file are load average figures giving the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 1, 5, and 15 minutes. An idle computer has a load number of 0 while each process using or waiting for CPU (the ready queue or run queue) increments the load number by 1. Any number more than 1 means there are some processes in waiting mode. For instance number 1 means there is no headroom and 1.7 means CPU is fully loaded and there are some process waiting for CPU that the amount of them is equal to 70% of CUP capability. On a multi-processor system, the load is relative to the number of processor cores available. The "100% utilization" mark is 1.00 on a single-core system, 2.00, on a dual-core, 4.00 on a quad-core, etc.[36]. A sample content of this file is as bellow:.

```
0.65 0.41 0.39 1/388 21697
```

Calculating the average of these numbers during an experiment and compare it to results from other experiments would be a good metric to decide about efficiency. This indicator shows which methode imposed more load to the system which may caused by I/O waiting or busy CPU.

Beside above indicators, the network traffic statistics were recorded from file */proc/net/dev*. This file contains information about the traffics to /from configured network interfaces. A sample content of the file and its information in as bellow:

```
root@controller:~# cat /proc/net/dev
Inter-|   Receive                                                |  Transmit
face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs
eth0: 12873909014 98231397    0    0    0     0          0  13334510 11978646753
eth1: 124534364627 164093861    0    5    0     0          0       351 49061761359
lo: 47817916731 126807309    0    0    0     0          0         0 47817916731
```

- bytes: The total number of bytes of data transmitted or received by the interface.

[9]http://en.wikipedia.org/wiki/Load_(computing)

- packets: The total number of packets of data transmitted or received by the interface.

- errs : The total number of transmit or receive errors detected by the device driver.

- drop : The total number of packets dropped by the device driver.

- fifo : The number of FIFO buffer errors.

- frame : The number of packet framing errors.

- colls : The number of collisions detected on the interface.

- compressed: The number of compressed packets transmitted or received by the device driver. (This appears to be unused in the 2.2.15 kernel.)

- carrier : The number of carrier losses detected by the device driver.

-multicast : The number of multicast frames transmitted or received by the device driver.

In order to further investigation on functionality and behaviour of networking service, all information related to each interface gathered. Actual bandwidth of an interface can be calculated through following formula:

$$Bandwidth(bps) = 8 \times \frac{(Total\ transmitted\ bytes)+(Total\ recieved\ bytes)}{(Total\ time\ of\ test)}$$

Moreover *power.pl* script frequently was gathering power cunsumption data by using *ipmitool* [71] and recorded them to another csv file. The ipmi that stands for *"Intelligent Platform Management Interface"* is a standardized computer system interface used for for out-of-band management of computer systems and monitoring of their operation [10]. The ipmi tool provides a simple command-line interface to IPMI-enabled devices through an IPMI LAN interface or Linux/Solaris kernel driver. In a linux system it can be installed through package installation or repositories.

The ipmitool can be used either locally on the target system or remotely from another system to gather data from sensors. It can be used to monitor many different sensors of the system including Power sensors, Fans, CPU temperature, etc. Following two samples shows the use of ipmitool tool locally and remotely:

[10]http://docs.oracle.com/cd/E19569-01/820-1188-12/core_ilom_ipmi.html

```
root@compute02:~#ipmitool -I open sensor get "Power Meter" "Power Supply1" "Power Supply2"
Locating sensor record...
Sensor ID              : Power Meter (0x39)
 Entity ID             : 7.10
 Sensor Type (Analog)  : Current
 Sensor Reading        : 52 (+/- 0) Watts
 Status                : Lower Critical
 Lower Non-Recoverable : na
 Lower Critical        : na
 Lower Non-Critical    : na
 Upper Non-Critical    : na
 Upper Critical        : na
 Upper Non-Recoverable : na

Sensor ID              : Power Supply 1 (0x3)
 Entity ID             : 10.1
 Sensor Type (Analog)  : Power Supply
 Sensor Reading        : 50 (+/- 0) Watts
 Status                : Lower Non-Critical
 Lower Non-Recoverable : na
 Lower Critical        : na
 Lower Non-Critical    : na
 Upper Non-Critical    : na
 Upper Non-Recoverable : na

Sensor ID              : Power Supply 2 (0x4)
 Entity ID             : 10.2
 Sensor Type (Analog)  : Power Supply
 Sensor Reading        : 35 (+/- 0) Watts
 Status                : Lower Non-Critical
 Lower Non-Recoverable : na
 Lower Critical        : na
 Lower Non-Critical    : na
 Upper Non-Critical    : na
 Upper Non-Recoverable : na
```

```
root@compute02:~# ipmitool -I lanplus -H 172.16.0.13 -U root -P changeme sensor get
 "Power Meter" "Power Supply 1" "Power Supply 2"

Locating sensor record...
Sensor ID              : Power Meter (0x39)
 Entity ID             : 7.10
 Sensor Type (Analog)  : Current
 Sensor Reading        : 56 (+/- 0) Watts
 Status                : Lower Critical
 Lower Non-Recoverable : na
 Lower Critical        : na
 Lower Non-Critical    : na
 Upper Non-Critical    : na
 Upper Critical        : na
 Upper Non-Recoverable : na

Sensor ID              : Power Supply 1 (0x3)
 Entity ID             : 10.1
 Sensor Type (Analog)  : Power Supply
 Sensor Reading        : 40 (+/- 0) Watts
 Status                : Lower Non-Critical
```

```
 Lower Non-Recoverable : na
 Lower Critical        : na
 Lower Non-Critical    : na
 Upper Non-Critical    : na
 Upper Critical        : na
 Upper Non-Recoverable : na

Sensor ID              : Power Supply 2 (0x4)
 Entity ID             : 10.2
 Sensor Type (Analog)  : Power Supply
 Sensor Reading        : 35 (+/- 0) Watts
 Status                : Lower Non-Critical
 Lower Non-Recoverable : na
 Lower Critical        : na
 Lower Non-Critical    : na
 Upper Non-Critical    : na
 Upper Critical        : na
 Upper Non-Recoverable : na
```

The average power/energy (*Power* $\times$ *time*) consumption of the system during each experiment can be calculated from these recorded data. It would be a good indicator to measure efficiency of each method. For example a suitable metric that would be used for more precise measuring efficiency is Consumed Energy for transferring amount of data($\frac{WattHour}{Mbyte}$)[11] or ($\frac{Joule}{Data}$) [42]. It means how much Energy is used by each method to transfer an amount of data [12].

The load monitoring tool were run in all physical and virtual machines during experiments and power monitoring tool were run on the compute nodes only. Stored data in the files is used to analyze the behavior of the system and measure the throughputs. Performance and efficiency of different configurations can be evaluated from analyzing these data.

To analyze these data, the average ($\bar{x}$) of each above mentioned parameter is calculated for each single test. These averages can be used to analyze the behavior of system during an experiments. Moreover each experiment has a set of averages from tests. According to *Central Limit Theorem* if tests repeat 30 times or more, the distribution of averages should be a *Normal Distribution* with Mean of $\mu_{\bar{x}}$ (Mean of Means) and standard deviation of $\sigma_{\bar{x}}$. Comparing these $\mu_{\bar{x}}$s together would indicate the experiment with better output (performance). For instance lowest $\mu_{\bar{x}}$ of power consumption and highest in bandwidth both are desired. Furthermore the

[11] http://www.bristol.ac.uk/environment/green_event_toolkit/footprinting.html
[12] http://www.rttonline.com/tt/TT2010_008.pdf

$\sigma_{\bar{x}}$ would show the more stable experiment so that lower standard deviation can be interpreted as more stability during the experiment.

The other group of data are the data gathered from migration tests. This group is included by times and network traffic data, recorded during live migration with SR-IOV and proposed method of this study. Analyzing these data shows downtime of network while using SR-IOV and the result of using the solution to prevent network loss during migration.

Chapter 4

Results

This chapter presents short description of system setup and functionality of written scripts, as well as a summarized content of log files and outcome of the experiments. It also includes description of implementation of proposed method and its result.

As described in section 3.2, and subsection 3.2.2, a test bed have been deployed and configured over the project equipments. This was the first task achieved and provided a suitable environment to implement the rest of project. Next each of mentioned methods in section 2.4 implemented separately and all planned experiments in section 3.3.2 were done successfully. At the end the proposed method of combining SR-IOV with bond driver and utilization in cloud environment were conducted.

4.1 System Setup

At the time of starting this project the latest release of OpenStack was HAVANA [72] (Also refer to table 2.1). This release which is the 8th release of OpenStack was chosen for this study. To meet the basic requirements of deploying a cloud, infrastructure was prepared according to figure 3.2, and *Ubuntu 12.04 LTS*(3.8.0-37-generic) has been used as operating system running on all those physical machines.

Following table 4.1 shows the roles and informations about physical machines in the cloud.

After basic preparation of infrastructure including OS installation, Network configuration and time synchronization (using NTP), three

Table 4.1: Cloud servers information

Role	Service		Network	
	Name	**Type**	**Name**	**IP**
Controller	Nova	Compute	External	192.168.200.231/22
	Glance	Image		
	Keystone	Identity		
	Cinder	Volume	Managment	192.168.10.1/24
	RabbitMQ	Messaging		
Networking	Neutron	Networking	External	192.168.200.230/22
			Managment	192.168.10.4/24
			Data	10.0.0.1/24
Compute node 1	Nova-Compute	Virtualization	Managment	192.168.10.2/24
		Compute	Data	10.0.0.2/24
			Experiment	10.1.1.1/24
Compute Node 2	Nova-Compute	Virtualization	Managment	192.168.10.3/24
		Compute	Data	10.0.0.3/24
			Experiment	10.1.1.2/24

common steps had to be done in all nodes preparing them to being cloud nods.

1. Installing Mysql server/client and configure it
2. Installing *python-software-properties* and adding cloud repositories
3. Installing Openstack client packages (for API calls in different nodes)

4.1.1 Controller node

In the controller node sequentially the following services had been installed and configured (including creation of related database). Related configuration files are available at appendix A:

Keystone as the identity service to defining needed users, tenants and roles. Sample of commands used to define user, tenant and role are as follow:

Sample Keystone commands

```
keystone tenant-create --name=admin --description="Admin Tenant"
keystone user-create --name=admin --pass=ADMIN_PASS --email=admin@example.com
keystone role-create --name=admin
keystone user-role-add --user=admin --tenant=admin --role=admin
```

Since there was no need to create multiple tenants in this project, only a tenant for admin and another for services were created. Also except the admin user and users for services no additional user were created.

Glance as the image service for registering virtual disk images. It is used either to add new images or take a snapshot of an image from an existing server for immediate storage. A sample glance command used in this project to create an AMI (Amazon Machine Image) is as follow:

Sample Glance command

```
glance image-create --name="centos-sriov_support" --is-public=true --disk-format=ami
--container-format=ami --file disk --property kernel_id=ab988570-5de9-4022-9bf9-153f
--property ramdisk_id=d9385661-7013-48c1-b786-86df79b3744f
```

A few number of linux images were created in this project to test and investigate suitable one for rest of project. Finally an AMI image of centos-6.4-x86_64 (kernel 2.6.32-358.18.1.el6.x86_64) was chosen and used to create VMs for experiments. This image was manipulated with adding specific SR-IOV driver. This will be explained in more detail in following sections.

Nova (including nova-api, nova-cert, nova-conductor, nova-consoleauth, nova-novncproxy, nova-scheduler) as the cloud computing fabric controller. This service is the main part of an IaaS system and was used to host and manage cloud computing systems. The configuration process of this service was based on multiple configuration files (appendix A.1). Some samples of nova commands, used in this environment after installation is as follow:

Sample Nova commands and results

```
Command to Create a new vm:
root@controller:~# nova boot --image centos --flavor 3 --key_name keynet\
--nic net-id=113847-875f-49d2-bbfe-44f3137  --availability-zone nova:compute02 vm7

Command to list vms:
root@controller:~# nova list --fields name,host,networks,status
+--------------------------+------+-----------+---------------------+--------+
| ID                       | Name | Host      | Networks            | Status |
+--------------------------+------+-----------+---------------------+--------+
| 16c96268-29e2-4d85-bd68 | vm1  | compute01 | internal=30.30.30.3 | ACTIVE |
| de88fbdb-a29e-49d5-b806 | vm2  | compute01 | internal=30.30.30.4 | ACTIVE |
```

```
| a1e12d26-767c-4e9e-be24 | vm3  | compute01 | internal=30.30.30.5 | ACTIVE |
| 442d3974-b795-4e7f-b8ee | vm4  | compute02 | internal=30.30.30.6 | ACTIVE |
| 646d55b1-1e40-474a-9a34 | vm5  | compute01 | internal=30.30.30.7 | ACTIVE |
| 8ebad85a-4155-41b0-b149 | vm6  | compute01 | internal=30.30.30.8 | ACTIVE |
| 301175c9-c992-4d12-8e0a | vm7  | compute02 | internal=30.30.30.9 | ACTIVE |
+-------------------------+------+-----------+---------------------+--------+

Command to migrate vm5 to compute node 02:
root@controller:~# nova live-migration vm5 compute02
```

Also the **RabbitMq** and **Cinder** were installed as messaging server and Block Storage Service on the controller. To have a better control and view from the whole cloud, the openstack **Horizon** (a graphical web interface) was installed as the dashboard of the cloud.

4.1.2 Network node

On a dedicated server the OpenStack **Neutron** was installed and configured to provide and manage software-defined networking for the cloud environment. The installation included *neutron-server, neutron-dhcp-agent, neutron-plugin-openvswitch-agent*, and *neutron-l3-agent*. Neutron configuration also is based on multiple configuration file which the major one is */etc/neutron/neutron.conf*. Configuration files of this setup are available at appendix A.2. Using Neutron, the needed Networks, Subnets and router were created. Figure 4.1 shows the virtual networks (External and Internal) and the virtual router of the cloud and their relations.

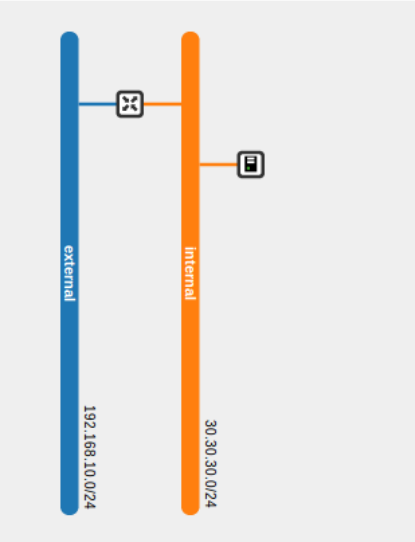


Figure 4.1: The virtual network schema of testbed cloud

Initially the GRE tunneling was chosen as the network type but later during experiments the configuration changed to flat-networking by Neutron. The reason was to provide the access to virtual networks by proposed method. This matter explained in more details in the discussion chapter.

4.1.3 Compute nodes

To support hosting virtual machines **Nova-Compute** service was installed on both compute nodes using *nova-compute-kvm* and *python-guestfs* packages. The default hypervisor of Openstack (KVM) was chosen to use in this setup.

Enabling SR-IOV VFs

To use SR-IOV Virtual Functions (VFs) it was needed to install the proper driver of Network Interface and configure it. Both compute nodes were equipped by Broadcom's NetXtreme II 10 Gigabit Ethernet controllers (BCM57810). The driver package downloaded, the loadable kernel module (*bnx2x*) created and the driver installed as follow:

Sample Nova commands and results

```
root@compute02:~# echo "options bnx2x num_vfs=8 > cat /etc/modprobe.d/bnx2x.conf
root@compute02:~# modprobe bnx2x num_vfs=8
root@compute02:~# update-initramfs -u
```

Above commands install the Broadcom SR-IOV drivers and activate 8 of VFs on the interface (the maximum number for this device on this kernel).

In addition to the Nova-compute packages, some neutron packages were needed to be installed on compute nodes to provide virtual networks for the VMs. Neutron packages included *neutron-plugin-openvswitch-agent* and *openvswitch-datapath-dkms*.

Important configuration files of compute nodes are available in appendix A.3

Enabling Live Migration

To enable live migration of virtual machines following reconfiguration were done on the nodes:

1. Unification of user id and group id for Nova
2. Setup a NFS server on controller node and and configure both compute nodes to use a shared directory to store VM images.
3. Editing QEMU and Libvirt configuration files (refer toA.3.1).

4.2 Investigational Experiments

According to section 3.1.2 an investigational study had to be done including all configurations.

4.2.1 Developped scripts

To run planned experiments, first of all couple of scripts were developed and tested. Written scripts, their functionality, and outputs are listed in following table:

Table 4.2: Developed scripts

Name	Function	Output
Load.pl	Recording operating system informations during experiment	Load_$hostname_$time.csv
Power.pl	Recording power consumption during experiment	Power_$hostname_$time.csv
RunExperiment.sh	Running experiments on different hosts and recording time	time.txt and run.log
Vm.sh	Create and preapering number of new vm for Experiment	
Analysis.pl	Analysing different recorded data according to methodology	$parameter.csv

1. **load.pl**(Appendix B.1.1)

   The script runs without getting any input and keeps running until the control value in the check file turns to 0. It retrieves the host name and creates a file as the output of monitoring. The file is named based on hostname and start time of recording to be identifiable afterwards. In an infinite loop with an interval of 0.5 second, the script opens different files under */proc*, finds target informations, retrieves

and records them into the output file. The retrieved data in each cycle are recorded in a comma separated line stamped by row number and time in milliseconds. Interval of 0.5 second and timestamp of milliseconds help for more accurate analysis.

This script successfully was used in all experiments running on all physical and virtual machines and worked as expected . A sample output of Load script is as bellow:

Sample output of load script

```
1,May,5,139930768534,4055044,3069572,666236,205604,0,0,2477,0,2010,288647,4430,5,20,
24,4,0,1157,0,797,143494,3399,5,13,3,0,1319,0,1213,145153,1030,0,11,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,32385103,24110,0,0,0,0,0,0,1426300,19095,0,0,0,0,0,0,18234108,0,
276274,0,0,0,0,0,0,11894225228,251690,0,0,0,0,0,0,0.00,0.00,0.00
```

2. **power.pl** (Appendix B.1.2) Independent from recording the system load information, the power scripts run to record power consumption information. Like load script, this script also designed to not get any input, running in an infinite loop and waiting for change in content of check file. It reads the output of ipmitool and put it to a comma separated line similar to load script. By some basic tests it was founded that the shortest response time of ipmitool is 1 second so the script was designed to call it in an interval of 1 second.

   This script was used in all experiments running on involved physical machines and worked successfully. Following a sample output of power script comes:

   Sample output of power script

   ```
   1,May,5,139930763633,54,45,40
   2,May,5,139930763733,54,45,40
   3,May,5,139930763833,54,45,40
   4,May,5,139930763934,54,45,40
   ```

3. **RunExperiment.sh** A short bash script was written to get the name of experiment from user, create VMs and needed directories, copy scripts to different machines, run scripts and start experiment in VM (using ssh). To run the experiment, this script calls another short script on the VM(s) to prevent affecting the results by overhead traffic and load of repeated ssh.

The script on the VM(s) calls iperf for 30 times and each time for 600 seconds. At the start of each cycle the script gets the time in millisecond and records it as the *START* time of test number N to a file called *time.txt*. At the end of the cycle again it gets the time, records it as the *END* time and waits for 60 seconds. The reason of this gap time is to let the indicators (especially power indicators) get back to normal status.

A sample of time file comes below:

Sample content of time file

```
emulate1500:
test 1 STARTED at 139793807807
test 1 STOPPED at 139793867812
test 2 STARTED at 139793873813
test 2 STOPPED at 139793933818
...

sriov1500:
test 1 STARTED at 139795053235
test 1 STOPPED at 139795113240
test 2 STARTED at 139795119241
test 2 STOPPED at 139795179246
...
```

4. **Analysis.pl**(Appendix B.1.3) Another script were written in perl and successfully used to explore huge output files and analyze data. This script gets the location of data file and time file, goes through both of them, finds the period of each test in an experiment and calculates considered values. To calculate different values for different indicators, this script uses all methods and formulas stated in section 3.3.4 in methodology chapter. Calculations for different experiments were recorded to separate csv files and average/mean of values from all experiments were recorded in some common csv files. These statistical files can be used by spreadsheets or statistical applications to analyze results of experiments. In this project since the main part of calculations were done by means of analysis script, the Microsoft Excel was used to plot graphs and further analysis.

following some sample outputs of this script come:

Sample analyzed outputs

```
Row,TestNumber,Mean of test,Standar Deviation of test,,iowait,,irq,,softirq,,Guest
1,1,19.3291772993371,4.29440145052113,,0.0736045437280861,,0.00129071363913591
,,0.400096484547187,,3.08801420125681
2,2,18.6970127741297,4.05551484122998,,0.091736748660701,,0.00211667202308436,,
1.36910011506522,,2.69863252899093
3,3,19.2283003748762,4.08185936575101,,0.0936564862775175,,0.00129393599448165,,
0.391865585777045,,3.06009046034024
...

Experiment,Mean of Means,Standard Deviation of Means,Average of Change in Buffer,
Average of Change in Cach,,Experiment,Mean of Means,Standard Deviation of Means,
Average of IOWait,Average of Steal,Average of Irq,Average of Guest,,Experiment,
Mean of Means,Standard Deviation of Means
emulate1500,1592106.58206072,295499.771568783,86.4,1041.86666666667,,
emulate1500,18.5579500315882,3.45214019934928,0.0855007515692534,
0.00146274012102198,0.547693606079757,2.9218652210868,,,emulate1500,
0.961727355700891,0.238797502705701
emulate9500,1609015.12719387,298644.056079123,79.7333333333333,1041.86666666667,,
emulate9500,18.6736213432234,3.46693340343597,0.0836239042418091,0.0011352346596
8727,0.204785975785968,3.62289059467063,,,emulate9500,1.09419086224555,0.26201735
7900165
```

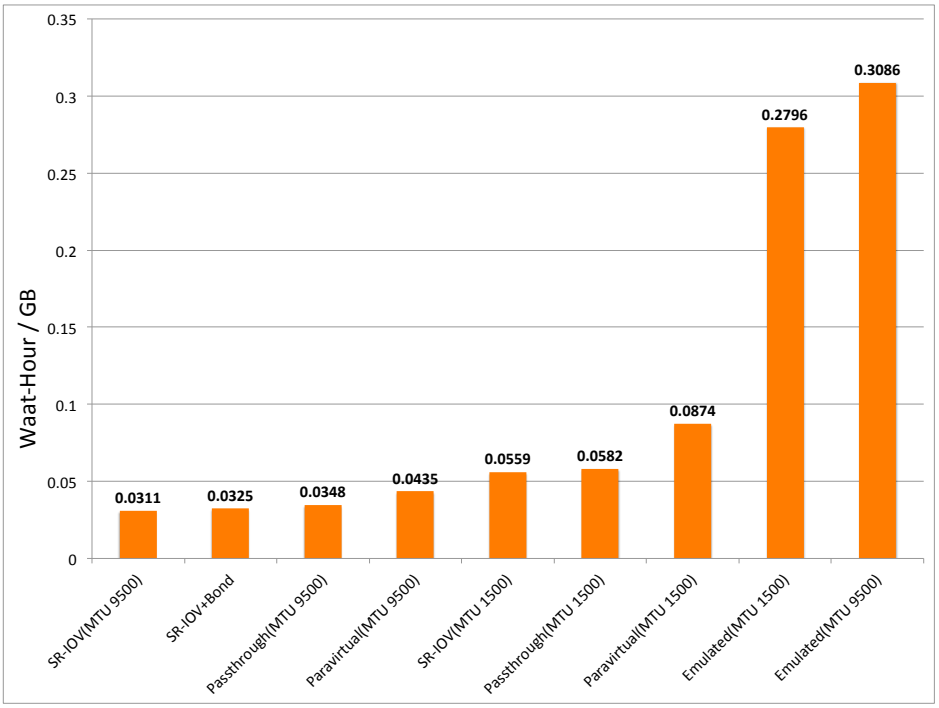


Figure 4.2: Sample graph (Energy consumption per data) plotted from output of Analysis script

4.2.2 Single VM experiments

After preparing environment and developing needed tools, investigational experiments were done. This phase of the project began by evaluating different methods on a single VM. Therefore couple of identical VMs (created from same image[1] with same general configurations and properties) were created on a same host and prepared for experiments.

The first compute node (Compute01) was chosen as the host of VMs and another compute node were configured as iperf server. According to the specification[2] of the Broadcom network interface (used in this project) the nominal bandwidth of the card is 10 Gb/s. Due to lack of high speed (10 Gbps) switch, to be able to achieve the maximum provided bandwidth , compute nodes were connected back-to-back on their SR-IOV capable NICs.

During experiments, in order to have more accurate evaluation and to prevent any intervention on the traffic or system load by others, only one VM was running on compute01 and no VM on compute02. To make sure the results can be replicated and are reliable, each experiment included 30 similar tests.

As the first experiment an emulated NIC was attached to the VM using libvirt with following commands and XML file:

Attaching emulated NIC to VM

```
#get experiment type from user
exp=$1
vm=vm_$exp
#creating VM
nova boot --image centos --flavor 3 --key_name keynet --nic net-id=113844d7-875f-49d2-bbfe
-44f3137ce4cd  --availability-zone nova:compute01 $vm
...
# retrieve instance name
instance=\$(nova list --fields name,instance_name | grep -i \$vm | awk '{print $6}')
# getting generated mac address by mac.sh script
mac=$(mac.sh)
...
# generating XML file for emulated device
echo -e "<interface type='direct'>
 <mac address='$mac'/>
 <source dev='eth5' mode='vepa'/>
 <model type='e1000'/>
 <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
```

[1] As mentioned above in same chapter, all VMs for experiments created from a CentOS-6.4-x86_64 (kernel 2.6.32-358.18.1.el6.x86_64) image.

[2] https://www.broadcom.com/collateral/pb/57810S-PB00-R.pdf

```
 </interface> " > $vm.xml

# attaching device to VM
virsh -c qemu+tcp://compute01/system attach-device \$instance $vm.xml;
```

In this configuration the 10 Gb NIC (eth5 in case of compute01) is emulated for VM. The fastest available mode for emulated device is e1000 that was used for generating XML file.

Three other experiments were done using similar procedure for Paravirtualized device, PCI Passthrough and SR-IOV. All experiment were successfully conducted according to the plan and all information recorded. The XMLs used to attach other devices are as following:

XMLs to attach other devices to VM

```
# generating XML file for paravirtualized device
echo -e "<interface type='bridge'>
<mac address='$mac'/>
<source bridge='exp-br'/>
<model type='virtio'/>
</interface>" > $vm.xml
...

 # generating XML file for passthrough device
echo -e " <hostdev mode='subsystem' type='pci' managed='yes'/>
<source/>
<address type='pci' domain='0x0000' bus='0x04' slot='0x00' function='0x1'/>
</source>
</hostdev>" > $vm.xml
...

# getting first available VF pci address
address=$(pci.sh)

# generating XML file for paravirtualized device
echo -e "<interface type='hostdev' managed='yes'>
<mac address='$mac'/>
<source>
$address
</source>
</interface>" > $vm.xml
```

For three above cases, there was no specific option about limiting or increasing bandwidth while attaching them to VM. To be able to attach paravirtualized NIC using eth5, the *exp-bridge* was created and connected only to eth5.

By a basic review and analysis on recorded data from first four experiments, It was observed that except emulated none of configurations reached or even were close to expected bandwidth. Following figure 4.3 illustrates the average bandwidth for those experiments.

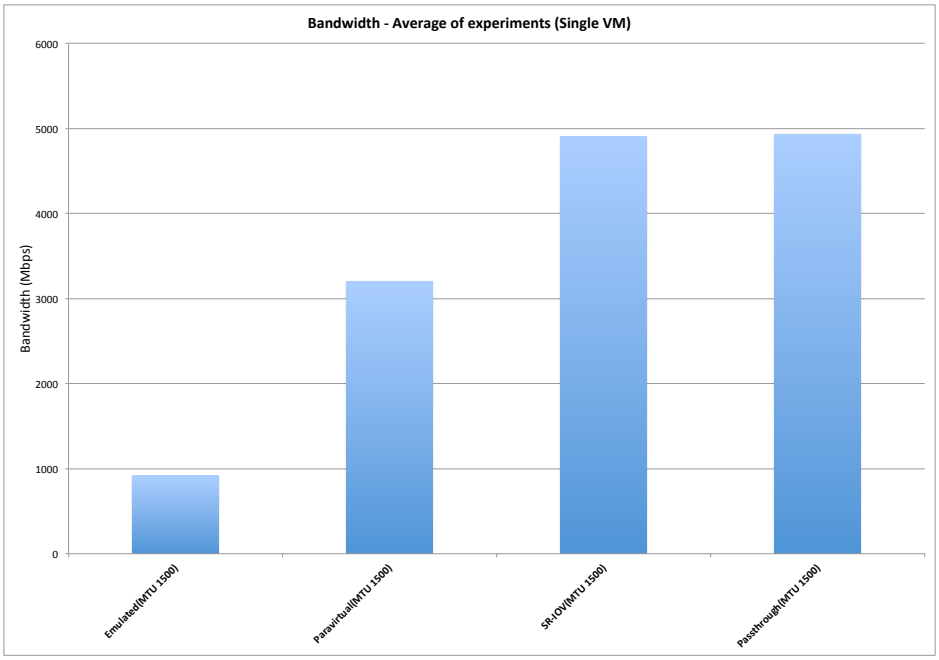


Figure 4.3: Average bandwidth of all methods during experiments of single VM with MTU 1500

To utilize maximum capability of the interface, use of *jumbo frames* were decided. Standard Ethernet Frames are 1518 bytes, including the MAC header and the CRC trailer. Jumbo frames, supported by many Gigabit NIC and switches, increase the size of an Ethernet frame to 9000 or 9500 bytes. [27]. To use the jumbo frames it was needed to reconfigure network interface on the VM by changing MTU[3] (Maximum transmission unit) to higher value. Some basic tests showed that the NIC used in this project supports the MTU of 9500 so this size was chosen for rest of experiments.

All previous experiments were repeated with MTU 9500. To get the benefits of jumbo frames, both local and destination network interfaces configured with this frame size. The raw outputs of new experiments clearly showed that the performances of three configurations including Paravirtualized, Passthrough and SR-IOV were increased significantly, but it needed more analysis on other factors to find out the overall impacts.

[3]http://en.wikipedia.org/wiki/Maximum_transmission_unit

The proposed method to utilize SR-IOV VFs is based on configuring bonding driver inside the VM. To understand the impacts of this method on networking performance and system loads, a combination of SR-IOV and paravirtualized interface was set up on a VM. This combination was handled by Linux bonding driver and same experiment was conducted. During the experiment the SR-IOV interface, as primary slave interface, always was active and paravirtualized interface as the slave was inactive (not interfering in network traffic). Achieved results from above experiments is as shown on figure 4.4.

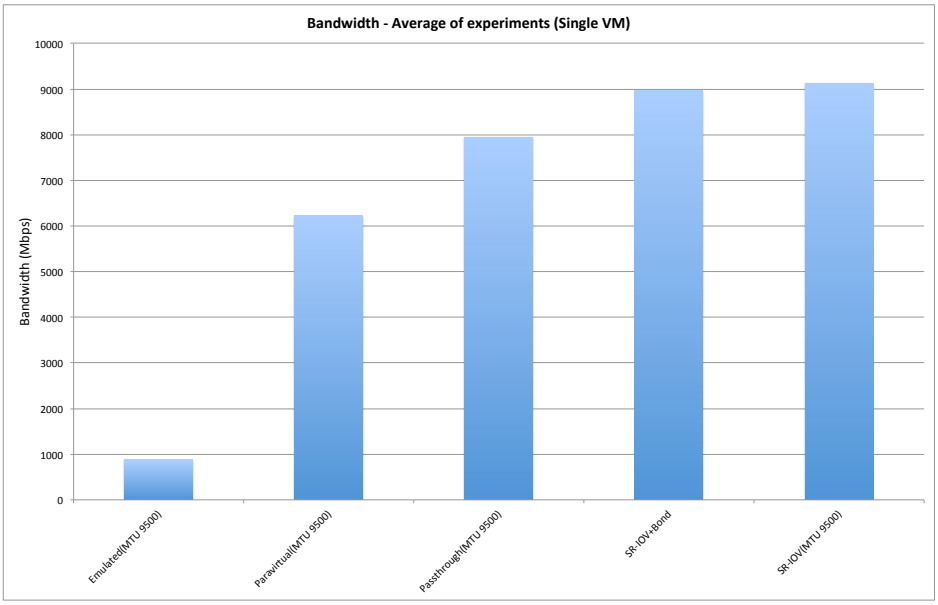


Figure 4.4: Average bandwidth of all methods during experiments of single VM with MTU 9500

This graph shows that changing the MTU to 9500, leaded to more performance increase in SR-IOV, comparing to pass through or paravirtualized. Moreover it can be seen that Bond configuration affected a bit on performance of SR-IOV due to its intervention to networking . The achieved bandwidth by SR-IOV was very close and even more than bare-metal (the physical host) bandwidth. To show this, an experiment of 30 tests was done to measure the bare-metal bandwidth on compute01. Following figure shows the comparison of achieved bandwidth in compute01 and a VM equipped with SR-IOV and SR-IOV in Bond interface.

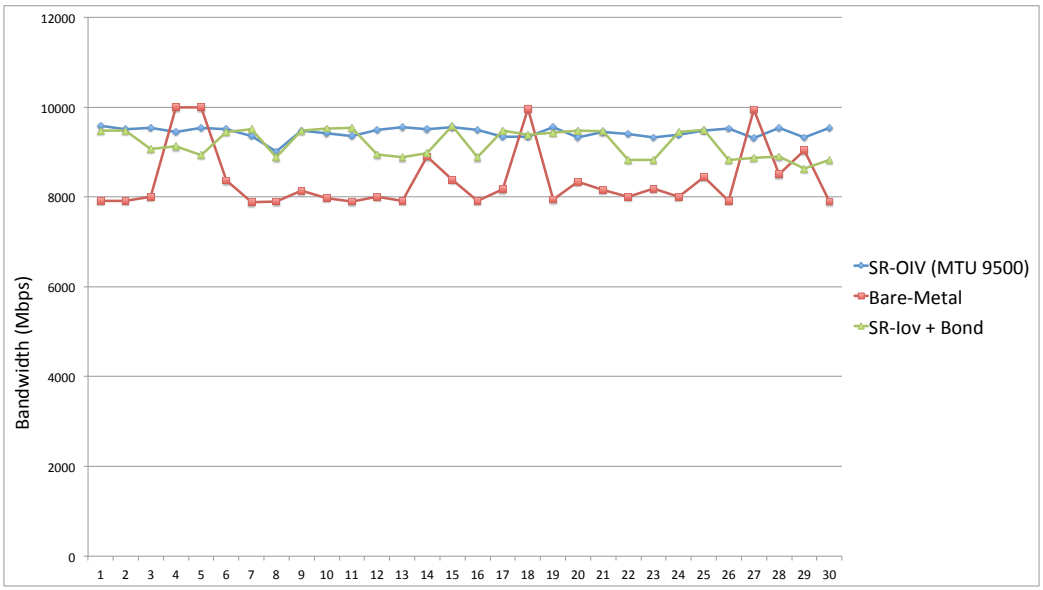


Figure 4.5: Comparing delivered bandwidth by SR-IOV configurations and the physical host

Early outcomes of recorded information from system loads showed that, however Passthrough and SR-IOV offer better networking performance, but they use more Memory. Also CPU usage information turned out that using jumbo packets impose some processing overheads to all configurations. This is considerably higher in case of paravirtualized. Following graphs shows these outcomes which need more analysis.

4.2.3 Multiple VMs experiments

As stated already, to observe the scalability of methods and investigating the impacts of increasing number of VMs (using specific method), some experiments had to be done. Due to limited number of interfaces on physical machines, practically the passthrough method is not scalable. Four more experiments were conducted using same tools as previous experiments but including 7 VMs. In each of experiments 7 similar VMs were configured to have similar properties and run the tests simultaneously. For all of these experiments, including Emulated, Paravirtualized, SR-IOV and SR-IOV(Bond), the MTU was set to 9500.

Outcomes showed that except Emulated device, VMs of other three configurations tried to use the maximum possible bandwidth. The sum of bandwidths achieved by each single VM is almost equal to maximum capability of the interface (10 Gbps). Following figure 4.6 illustrates the average bandwidth of all VMs during experiments.

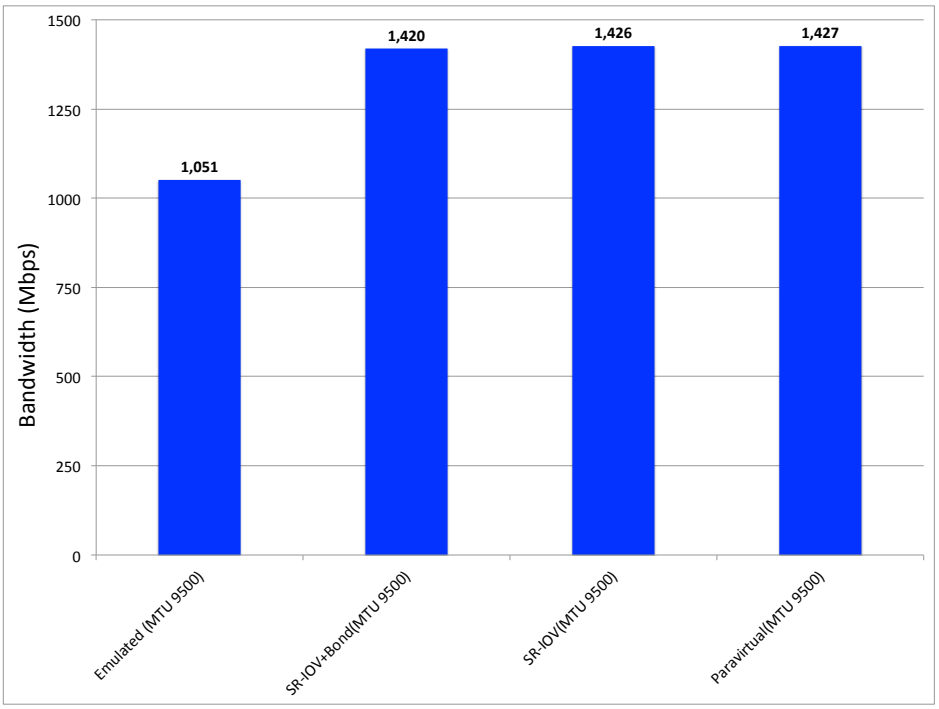


Figure 4.6: Average bandwidth of all methods with multiple VMs

4.2.4 Idle system measurement

To have more precise analysis and interpretation of all recorded data, and a better understanding about system conditions, a measurement from idle system conducted. This measurement aimed to monitor and record the same system indicators used for previous experiments, while system is idle. Idle means hosting no VM, having no activity of specific applications, and no networking traffic. For this purpose same monitoring script was running on both compute nodes for 24 hours to cover a whole day time.

4.3 Dynamic reconfiguration in SR-IOV enabled Openstack

Getting done the investigational experiments, observing early outcomes and analyzing the results, were convincing to continue project to implement the proposed method. In order to utilizing SR-IOV VFs in cloud instances and performing live migration following steps were carried out.

1. Developing supportive scripts
2. Editing Openstack codes to use supportive scripts

3. Reconfiguring Neutron

The plan was to attach a SR-IOV VF while creating a VM and configuring the bond interface in very first boot. For live migration it was needed to detach the VF before migration and reattach it immediately at the destination with same mac address. Using same mac address leads to having exactly the same interface at the destination, from VMs perspective.

4.3.1 Supportive scripts

In order to automate the process of attach, detach and reattaching the SR-IOV VFs to cloud instances, following short scripts were developed using shell scripting:

Table 4.3: Supportive scripts to handle use of SR-IOV VFs

Name	**Function**	**Output**
mac.sh	Generating a unique mac address	mac address
pci.sh	Finding the first available PCI address of VFs	pci address (in XML tag)
sriov.sh	Generating a XML file, Attach/Detach/Reattach VFs to VMs	Successful/ Error - vm.csv - log.txt
bond.sh	Configuring bonding driver on the VMs	

All of these scripts are available in appendix.

1. **mac.sh**: This script is developed to generate a standard unique mac addres. It searches the XML files of all instances in the cloud to check if the generated mac is not already in use and print it out.(B.2.1)

2. **pci.sh**: This script is developed to get the host name, retrive pci address of all active VFs on the host and find the first available (not in use)VF. The address is printed out in libvirt XML format. (B.2.2)

3. **sriov.sh**: The proccess of attach/dettach a SR-IOV VF to an instance includes creating a xml file containing a unique mac address and pci address (of the VF), and callin virsh (*attach-device/detach-device*) command. The sriov script is developed to get the instance name and command (i.e. Attach, Detach or Reattach) and perform the task based on the command. If the script is called to attach, it will call mac.sh and pci.sh, create the XML, attach the device, and record information. Those information contain instance name, mac address and pci address that are recorded to a file for further use. If the

command is detach the script will retrieve the mac address and pci address from the file, create XML and calls the libvirt to detach the VF. For reattaching, it retrives the mac address from the file and call the pci.sh for new pci address (at new host), attach the VF and update the information in the file. (B.2.3)

4. **bond.sh**: this script is written to be run inside the VMs, install bonding driver and configure network interface files. It can be injected while creating VMs[4] or be located in a customized cloud image. In this project the decision was using customized image file.

4.3.2 Enabling OpenStack to attach SR-IOV VF to VMs and perform Live Migration

To make OpenStack to use supportive scripts at the right time, two of python scripts should be edited.

driver.py (*/usr/share/pyshared/nova/virt/libvirt/driver.py*)
This script is one of the main scripts of nova-compute which handles many tasks. *driver* is responsible of creating instances and boot them in contribution with libvirt. To attach the Virtual Function to the VM at the right time, a part of driver.py was edited to call the siriov script. A piece of code was added after functions of creating image and launching instance , and before functions of final boot. The change was as bellow:

Attach VF at the time of creating VM

```
2090 self._create_domain_and_network(xml, instance, network_info,
2091                                 block_device_info, context=context)
2092 LOG.debug(_("Instance is running"), instance=instance)
2093
2094  def _wait_for_boot():
2095     """Called at an interval until the VM is running."""
2096     state = self.get_info(instance)['state']
2097
2098     if state == power_state.RUNNING:
2099        LOG.info(_("Instance spawned successfully."),
2100                  instance=instance)
2101        # Attaching SR-IOV VF starts here ##########
2102        import subprocess as sub
2103        LOG.critical("SR-IOV: " + '/var/lib/nova/share/sriov.sh
' + instance['name'] + ' attach')
2104        assign_sriov = sub.Popen(['/var/lib/nova/share/sriov.sh', instance['name'],
'attach'], stdout=sub.PIPE, stderr=sub.PIPE)
2105         result = assign_sriov.communicate()[0]
2106         LOG.critical("SR-IOV: " + result)
```

[4]Using –user-data or –file options of nova boot

```
2107         # Attaching SR-IOV VF ends here #############
2108         raise loopingcall.LoopingCallDone()
2109
2110   timer = loopingcall.FixedIntervalLoopingCall(_wait_for_boot)
2111    timer.start(interval=0.5).wait()
```

This script also is responsible for preparing VM for live migration when migration is initiated at origin. Thus to address the live migration issue, another part of *driver* was edited to call sriov.sh for detaching VF befor initiating the migration. The following part was edited:

Detach VF at the time of starting live migration

```
4086 # Do live migration.
4087        try:
4088            if block_migration:
4089                flaglist = CONF.block_migration_flag.split(',')
4090            else:
4091                flaglist = CONF.live_migration_flag.split(',')
4092            flagvals = [getattr(libvirt, x.strip()) for x in flaglist]
4093            logical_sum = reduce(lambda x, y: x | y, flagvals)
4094
4095            dom = self._lookup_by_name(instance["name"])
4096            ##SR-IOV: START DETACH FOR LIVE MIGRATION########
4097            import subprocess as sub
4098            LOG.critical("SR-IOV: " + '/var/lib/nova/instances/sriov/sriov.sh ' +
instance['name'] + ' detach')
4099            detach_sriov_for_livemigration = sub.Popen(['/var/lib/nova/instances/sriov
/sriov.sh', instance['name'], 'detach'], stdout=sub.PIPE, stderr=sub.PIPE)
4100            result, error_sub = detach_sriov_for_livemigration.communicate()
4101            LOG.critical("SR-IOVMigration: " + result)
4102            LOG.critical("SR-IOVMigration Error: " + error_sub)
4103            #SR-IOV: END DETACH FOR LIVE MIGRATION #########
4104            dom.migrateToURI(CONF.live_migration_uri \% dest,
4105                             logical_sum,
4106                             None,
4107                             CONF.live_migration_bandwidth)
4108
4109        except Exception as e:
4110            with excutils.save_and_reraise_exception():
4111                LOG.error(_("Live Migration failure:\%s"), e,
4112                          instance=instance)
4113                recover_method(context, instance, dest, block_migration)
4114
4115        # Waiting for completion of live_migration
```

manager.py (*/usr/share/pyshared/nova/compute/manager.py*)
At the destination, this script is responsible to finalize the migration, activating the instance and change the state of VM. The new VF should be attached immediately at destination. Thus a part of *manager*, before ending migration tasks, was edited to cal sriov.sh to reattach VF to VM.

Reattach VF at destination of live migration

```
4215       # Restore instance state
4216        current_power_state = self._get_power_state(context, instance)
4217        node_name = None
4218        try:
4219            compute_node = self._get_compute_info(context, self.host)
4220            node_name = compute_node['hypervisor_hostname']
4221        except exception.NotFound:
4222            LOG.exception(_('Failed to get compute_info for \%s') \% self.host)
4223        finally:
4224            instance = self._instance_update(context, instance['uuid'],
4225                    host=self.host, power_state=current_power_state,
4226                    vm_state=vm_states.ACTIVE, task_state=None,
4227                    expected_task_state=task_states.MIGRATING,
4228                    node=node_name)
4229
4230        # NOTE(vish): this is necessary to update dhcp
4231        self.network_api.setup_networks_on_host(context, instance, self.host)
4232        # SR-IOV: START REATTACH POST LIVE MIGRATION
4233        import subprocess as sub
4234        LOG.critical("Vangelis: " + '/var/lib/nova/instances/sriov/sriov.sh ' +
 instance['name'] + ' reattach')
4235        detach_sriov_for_livemigration = sub.Popen(['/var/lib/nova/instances/sriov/
sriov.sh', instance['name'], 'reattach'], stdout=sub.PIPE, stderr=sub.PIPE)
4236        result, error_sub = detach_sriov_for_livemigration.communicate()
4237        LOG.critical("VangelisMigration: " + result)
4238        LOG.critical("VangelisMigration Error: " + error_sub)
4239        #SR-IOV: ENDREATTACH POST LIVE MIGRATION
4240        self._notify_about_instance_usage(
4241                     context, instance, "live_migration.post.dest.end",
4242                     network_info=network_info)
```

Figure 4.7 illustrates the flow of procedures to utilize SR-IOV VFs in Openstack cloud. The whole procedure were implemented and tested successfully.

4.3.3 Conducting and Evaluating the Live Migration

To evaluate the live migration with SR-IOV two experiments were conducted. At the first experiments for several times a VM with only SR-IOV interface (no bonding) migrated with nova (from Compute01 to Compute02 and vice versa) to measure the average of network down times. This VM originally was created in Compute01 and a heavy network traffic was established between this VM and one in Compute02. In the second experiment a similar VM with bonding driver was migrated several times to measure the reduction of performance and probable network loss.

To have an accurate evaluation, it was needed to record the exact time

of different actions during these tests. To be able to record needed times some lines of code added to supportive scripts to record the certain times of calling the script by Nova and performing the actions (Attach, Detach or Reattach) in millisecond. Also the information about network interfaces were read from */proc/net/dev* and recorded to a file in an interval of 0.1 second.

It should be noticed that running second experiment (Migration with bonding configuration) encountered a technical issue regarding to incompatibility of O.S. kernel and SR-IOV driver. This matter is explained in detail in a section (6.3.3) in discussion chapter.

Except above mentioned issue, the live migration test was successful and according to assumption. VM was migrated properly and all of functions performed in a timely manner.

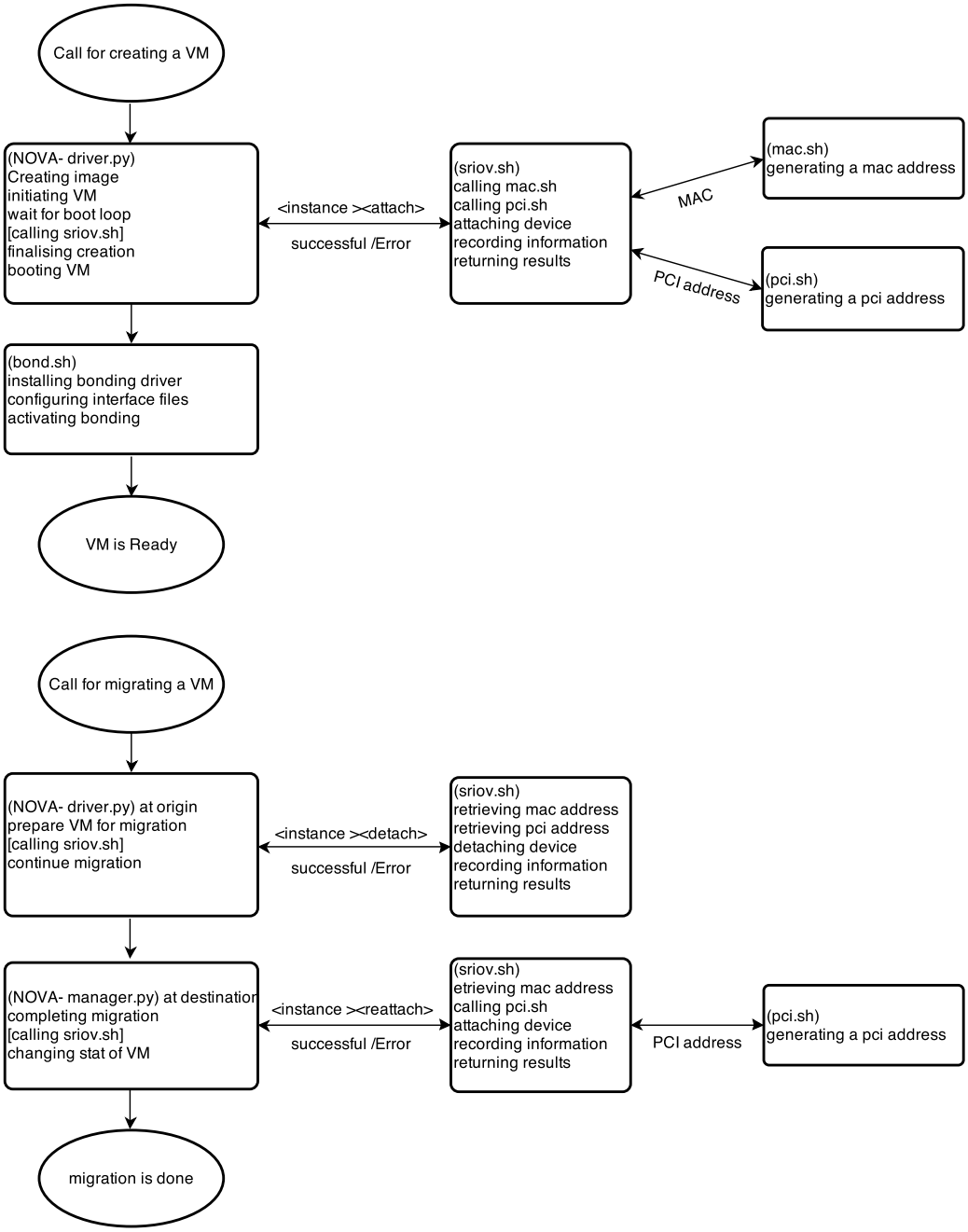


Figure 4.7: The flow of handling SR-IOV usage in Openstack

Chapter 5

Analysis

In the analysis chapter a comprehensive evaluation of the results will take place. The results of both investigational experiments and implementation of proposed method for conducting live-migration with SR-IOV, are explained in detail and analysed. The positive impact of using ethernet SR-IOV in efficiency and performance of the networking and the ability of dynamic reconfigurations are also displayed.

5.1 Evaluation of different methods

The experimental results of the previous section clearly demonstrate the impacts of utilizing different methods of I/O virtualization on network performance. The early outcomes show the increase of throughput while a direct access to I/O device. In case of this project passthrough and SR-IOV techniques offered significantly higher performance. They even exceeded the bare-metal bandwidth. To have better understanding of impacts of different configurations, following analysis were done on the results.

5.1.1 Different methods with Single VM

In this part, obtained results from experiments with a single Virtual Machine are analyzed.

Considering only the bandwidth, by default MTU, Passthrough and SR-IOV exposed almost equal performance that was considerably more than two other methods. Both of them went beyond the bare metal bandwidth while Paravirtualized was very close to it. Figure 5.1 shows the average of delivered bandwidth by each method during experiments.

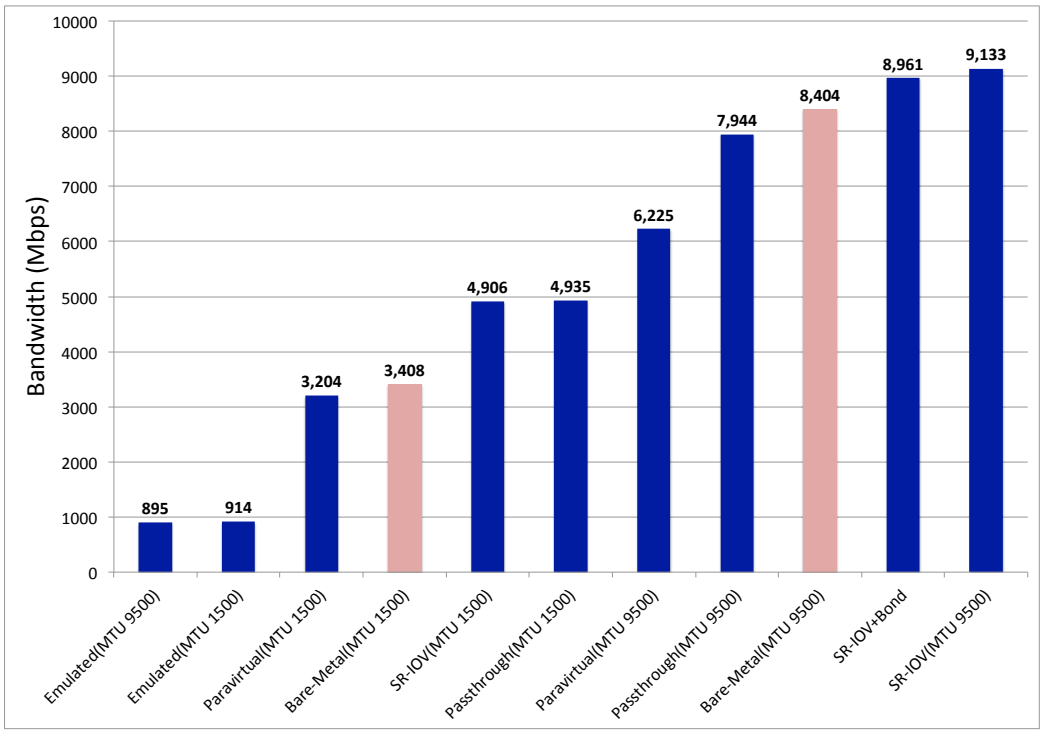


Figure 5.1: Bandwidth- Average of all experiments with single VM

This graph shows that with available options in current virtualization technology, even with powerful infrastructure, the emulation technique is unable to deliver very high throughput. Changing the MTU also did not improve the throughput of emulated device, while this action had significant effect on other methods. Changing MTU from 1500 to 9500 led to increase of bandwidth by 94% in Paravirtualized, 86% in SR-IOV, and 61% in Passthrough (in average). This change enabled the SR-IOV to deliver the highest throughput between all methods (9,133 Mbps).

Also it can be seen that utilizing the Bonding driver had a little impact (in average 172 Mbps out of 9133 Mbps) on the throughput of SR-IOV. According to implementation of linux bonding driver and its features[4], this little deduction of bandwidth is expected. Since in this configuration the *Active-backup* mode of bonding is used, the whole traffic is passed only through one interface (Active slave). It means no additional network interface as a support (for passing the traffic simultaneously with main interface) is aggregated in configuration. The bonding driver has some intervention on networking in order to traffic handling and *status detection* that caused this few decrement.

Figure 5.2 illustrates the behaviour of each method during the experiments.

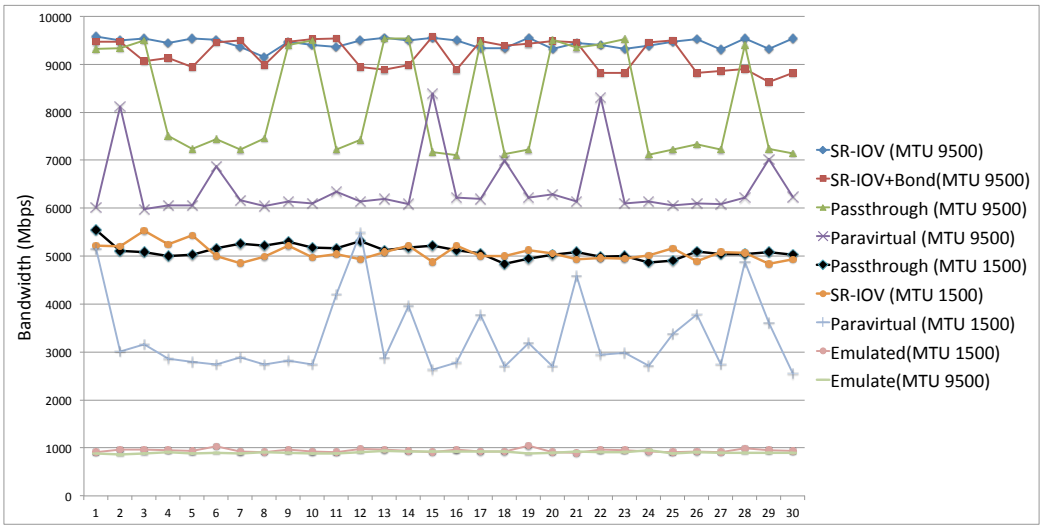


Figure 5.2: Bandwidth- Average of each test during experiments with single VM

From this graph, it appears that the Emulation technique had more stable behavior than other configurations. With MTU of 1500 except Paravirtualized, other methods showed almost a stable behavior. But increasing the MTU caused a turbulence in Passthrough technique as well. To have more accurate analysis on stability (relative stability) of methods, the distribution of average bandwidth by each method is plotted. This distribution is based on mean and standard deviation of averages ($\mu_{\bar{x}}$ and $\sigma_{\bar{x}}$).

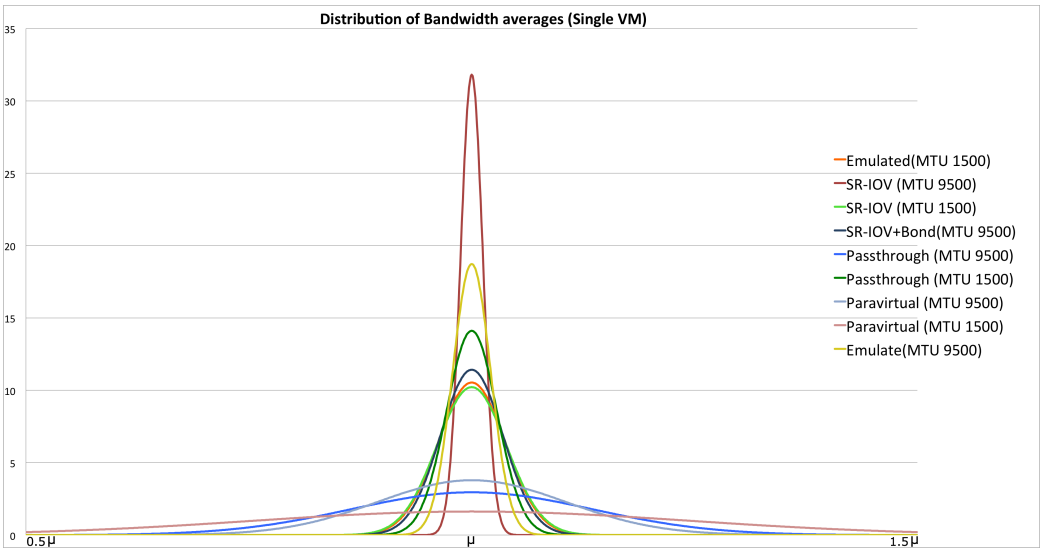


Figure 5.3: Bandwidth- Distribution of test averages during experiments with single VM based on Mean and Standard deviation of averages

This plot clearly shows that based on mean, standard deviation and the scale of bandwidth averages, the SR-IOV (MTU 9500) was the most stable and Paravirtualized (MTU 1500) was the least stable configurations. However the Passthrough with default MTU was very stable but increasing the frame size and amount of transferred data made it unstable. This strange behaviour can be analyzed based on other system indicators such as IRQ that is described below in this chapter. Both Emulation and Paravirtualization techniques are highly dependent to the VMM and therefore to the host activities and loads. This matter directly affects their functionality and throughput. The delivered bandwidth and amount of transferred data by emulation is much less than paravirtualization, especially when it comes to use of jumbo frames. This matter caused less intervention by the VMM due to lower IRQ and overly less CPU activity for emulation. That could be the reason to show more stable behaviour by emulated device than Paravirtualized in same situations. This will be presented in more detail in following parts.

Only based on bandwidth analysis, SR-IOV was the best configuration due to offering highest bandwidth and being very stable. Although Emulation was one of the most stable configurations, it offered the lowest bandwidth and no improvement by increasing frame size. These points altogether made it the worst option from point of networking performance.

To understand the mutual impacts of network throughput and system load on each other and evaluating the efficiency of each method, the outcomes of load monitoring were analyzed.

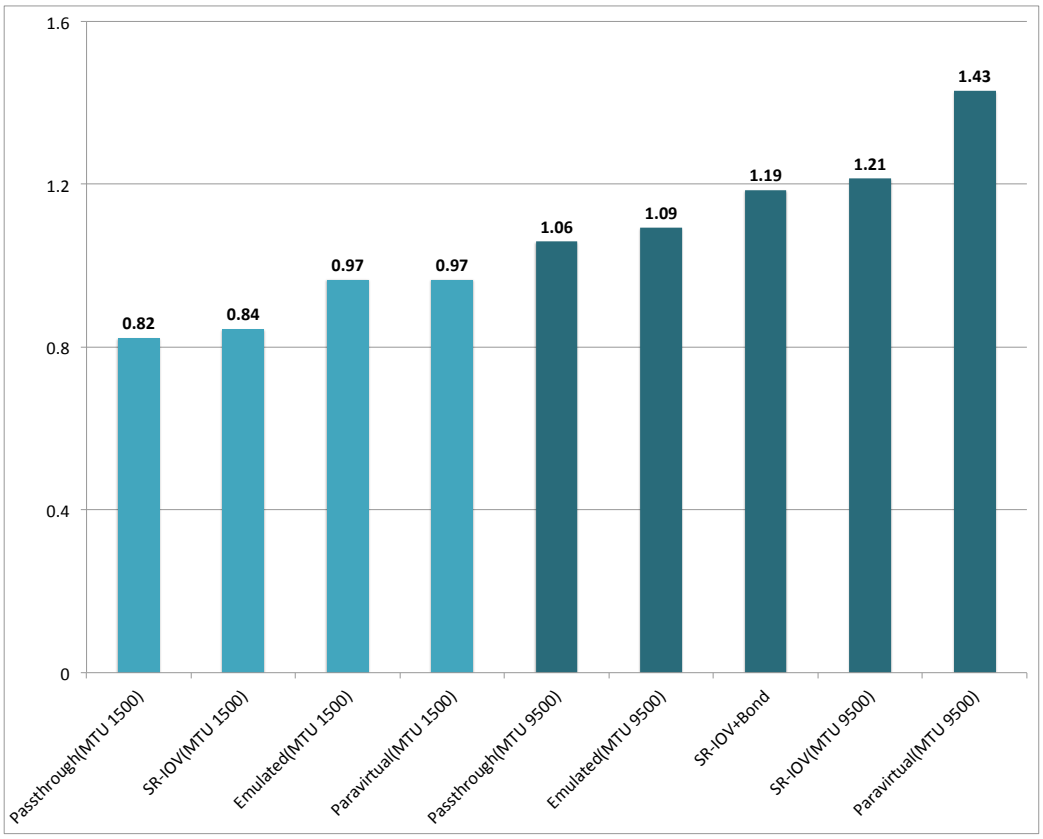


Figure 5.4: System load in host (Compute1)- Average of all experiments with single VM

Figure 5.4 illustrates the load average in compute node1 (Host of VM) during different experiments. This graph presents the level of impacts on host resources by utilizing each of techniques. It can be obtained that direct access techniques impose less load to the host system as it was expected. By the default frame size, while Passthrough showed lowest load average, Emulation and Paravirtualization had exactly equal value that was 1.2 times more than Passthrough. Increasing the MTU to 9500, which led to have higher throughputs, affected the system load by increase of 30% in Passthrough, 43% in SR-IOV and 47% in Paravirtualized. Except in case of Emulation, this increment exactly follows the same pattern as increment in bandwidth.

Load raise by Emulation was 12% while its throughput did not increase. This is due to limits in technology and implementation of Emulation (e1000), that its throughput does not exceed the 1 Gbps. But the amount of used memory increased to handle bigger frame size and it affected the value of system load.

In both case of MTU 1500 and 9500 the system load by SR-IOV is almost 85% of system load by Paravirtualization while it always offers higher bandwidth than Paravirtualized.

CPU load and its affecting parameters can describe a big part of above matter. Following figures 5.5 shows the average of total CPU usage in different experiments and the share of affecting parameters. All of these values are calculated base on methods stated in section 3.3.4. It should be mentioned that the total CPU usage includes the IRQ and Guest.

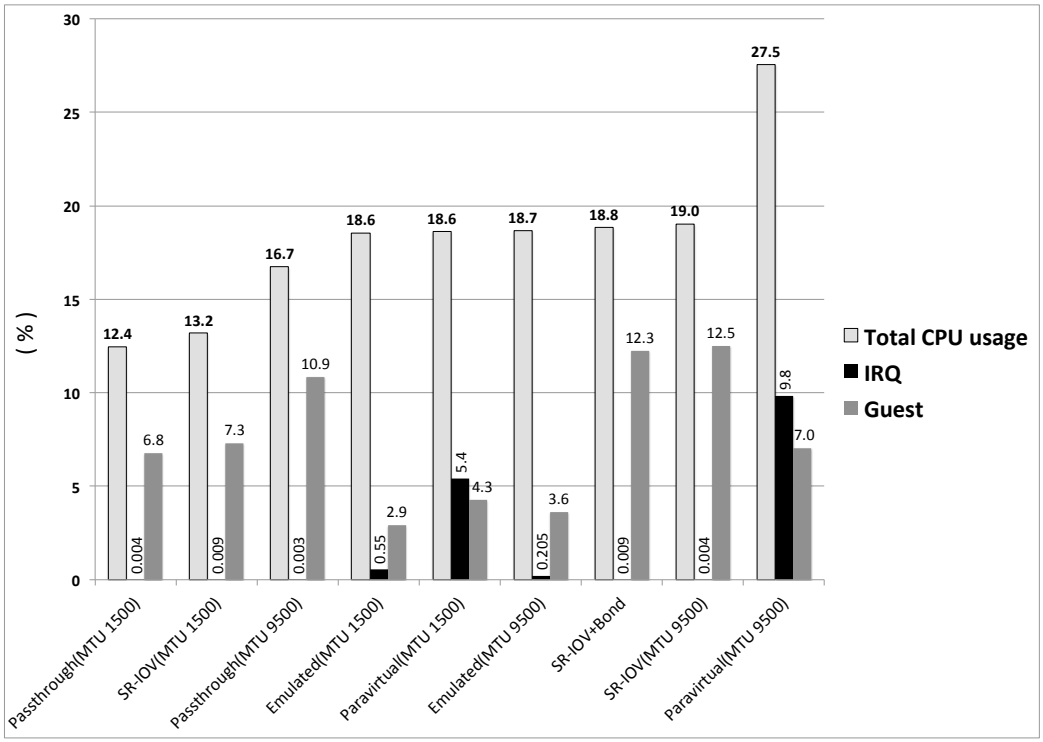


Figure 5.5: CPU usage in host (Compute1)- Average of experiments with single VM

The first obvious point is very high level of interrupt request by Paravirtualized technique both with MTU 1500 and MTU 9500. This high level of IRQ in the host CPU would have a considerable negative impact on CPU performance. By changing frame size, CPU usage in this configuration increased by 47% but the IRQ increased by 81% and took almos 10% of CPU time.

However the share of guest processes in both emulated and Paravirtualized is lower than SR-IOV and Passthrough[1], but interrupt request and

[1]This is due to network performance. Since the traffic comes from within the VM and

system processes have higher shares in CPU usage. Except the load of experimental tools (monitoring scripts) which was equal for all of the experiments, there was no considerable process in user mode. This overhead was calculated from results of idle measurement (refer to section 4.2.4) and it was almost 1.2% of CPU usage in average. It means that the rest of value is the share of System processes. Following table shows these amounts for all of the experiments.

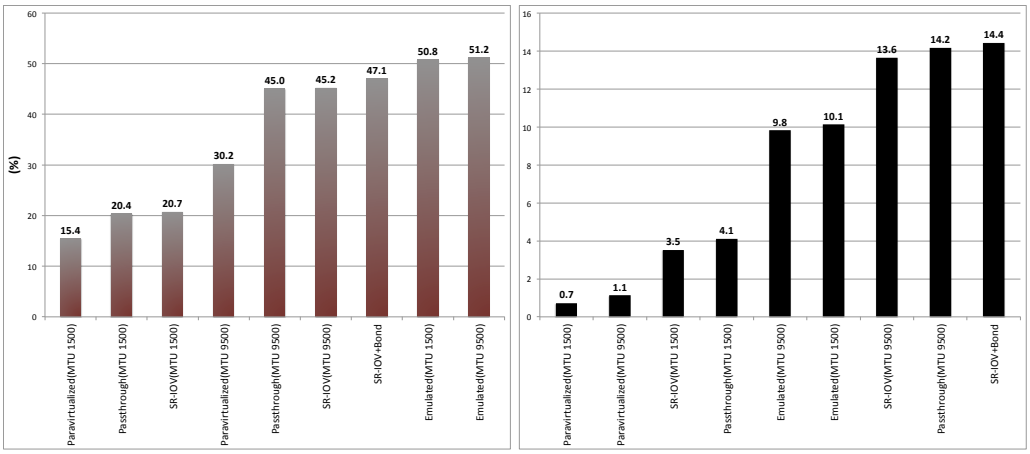
Table 5.1: Share of different parameters in CPU usage (Compute1), Average of experiments with single VM

Experiment	**MTU**	**Total CPU usage %**	**Guest %**	**irq %**	**Rest of processes %**
Passthrough	1500	12.4	6.8	0.004	**5.6**
	9500	16.7	10.9	0.003	**5.8**
SR-IOV	1500	13.2	7.3	0.009	**5.9**
	9500	19	12.5	0.004	**6.5**
	9500 Bond	18.8	12.3	0.009	**6.5**
Paravirtualized	1500	18.6	4.3	5.4	**8.9**
	9500	27	7	9.8	**10.2**
Emulated	1500	18.6	2.9	0.6	**15.1**
	9500	18.7	3.6	0.2	**14.9**

This is obvious that there were more kernel processes while utilizing Emulated and Paravirtualized device. It is according to assumption since both techniques need the hypervisor intervention to provide networking for VM. Emulation imposed more kernel processes to the system and delivered much less bandwidth.

Taking a look to inside the VM and its processing activities showed that the behaviour of all configurations were according to the assumptions. Figures 5.6a and 5.6b illustrate the CPU usage and IRQ percentage inside the Virtual Machine.

the VM is using 10Gbps, it should normally consume more CPU cycles inside the VM.



(a) CPU usage inside VM- Average of usage during experiments with single VM

(b) Interrupt request inside VM - Average of experiments with single VM

Referring to section 2.4 and figure 2.6 (in Background chapter), these plots show that working with real/physical drivers imposes more CPU load to the VM. This is due to matter of handling the device and interrupt requests inside the VM (Virtual interrupts). In Paravirtualization this task is shared between VM and VMM.

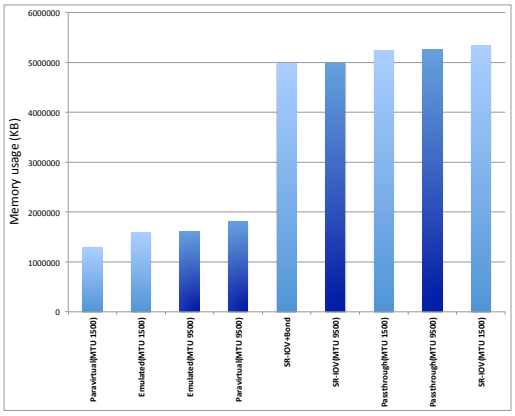
To have a proper analysis of these graphs two points should be considered: First, the processing resources inside the VM was one-four of physical host. It means the level of CPU usage in the VM should not be directly compare with levels in host. Second, a big share of processing load inside the VMs was imposed by *iperf* to generate and send huge amount of data. This processing load was different from one experiment to another based on the amount of generated and transferred data. It is more visible in compute node 2, where except handling the traffic, system had no overhead (Refer to figure C.1 in appendix C.1). Recorded information from Compute02 shows that above mentioned overhead is a combination of amount of data and stability of the networking.

Inside the VM, SR-IOV and Passthrough showed same behaviour in both case of MTU. The percentage of increase in CPU usage by them (125%) was more than the increase by Paravirtual (96%). This is totally according to the their throughput which means generating more data imposed more load. But the behavior of Emulated device was totally different. Having lower IRQ than SR-IOV, it had more CPU usage that is due to matter of interrupt handling in this technique. Trapping and emulating IRQ and register r/w by the VMM and injection of virtual IRQ to the

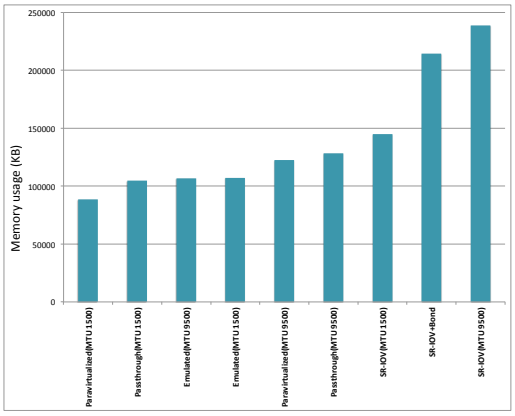
VM affected its processing performance. These graphs also show that the concept of Virtual Function (VF) improved the matter of interrupt handling in direct access method. After increasing the frame size, with higher bandwidth still SR-IOV had less percentage of IRQ. This is one of the reasons impact the stability of passthrough by increasing frame size. Moreover it can be seen that adding a bond interface to the VM imposed a little more IRQ and CPU usage to the VM which is due to bonding driver intervention and activities to handle the failover.

From the system load perspective, it appears that Paravirtualization method is the worst configuration, since it imposed highest load to the system among all configurations. But considering the performance inside the VM as well as data generation overhead and throughput of different methods, actually Emulation has lowest performance.

Another resource to consider is Memory. Figures 5.7a and 5.7b illustrate the average amount of memory, used during experiments in the compute node 1 and in the VM.



(a) Memory usage in Compute1 - Average of experiments with single VM

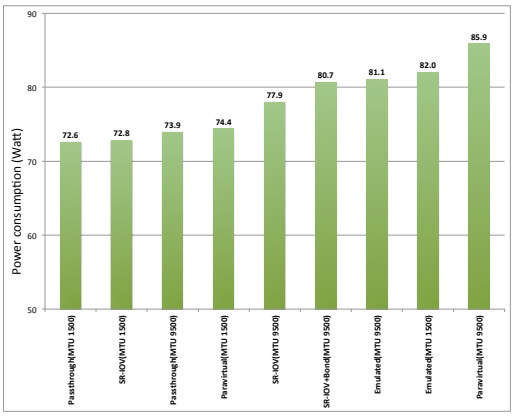


(b) Memory usage in VM- Average of experiments with single VM

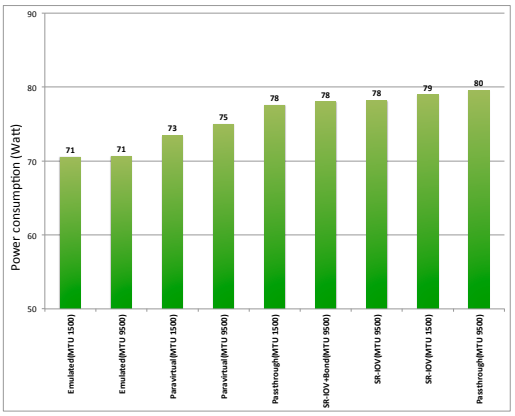
Recorded data from memory usage shows that, Paravirtualization technique benefits from a good memory management. However amount of memory to some extend is in accordance to iperf activity (to utilize the bandwidth as much as possible), but very high difference between Direct Access methods and Paravirtualization is noteworthy. Comparing to the host, the amount of memory inside the VM is not very considerable and iperf was using this memory (that is a part of user space in the host). More investigations in the whole system during different experiments showed that, the biggest share of memory is used by libvirt (the VMM library) This

also can be proved by recorded data from second compute node (refer to appendix C.2). Furthermore the utilization of Cache and Buffer by Para-virtualization was significantly more. This means that handling a direct access device and generating a big amount of data at the same time, had high impact on memory by other methods. Despite this, given the amount of work that was performed with SR-IOV or Passthrough, Memory might not be a bottleneck.

One of the most important parameters for evaluating the efficiency of the methods is the power consumption. Power by itself is just an indicator that shows the average of electricity power, consumed by physical hosts in a period of time. This amount is dependent on different parameters such as activity of CPU, Fans, Memory, Network cards, etc. Figures 5.8a and 5.8b show the average of power that was consumed during each experiment on both compute nodes.



(a) Power consumption in Compute1- Average of experiments with single VM



(b) Power consumption in Compute2- Average of experiments with single VM

For analysing these information it should be considered that the data were taken from physical sensors. Outputs of these sensors are not linear and changed stepwise. Also environmental conditions could affect on the whole experiment output. In case of this study the server room was not very professional (standard) or isolated and its temperature might affect the measurements.

Since there were no additional process activity and overheads on compute node 2 , to a large extend throughputs of power monitoring follows the expected pattern. It is based on the level of CPU and memory activity, where the cost of CPU usage is 2 times more than memory. The CPU us-

age in this node is based on the amount of transferred data (Bandwidth) and the stability of network. According to this pattern, Passthrough and SR-IOV configurations are placed on top of Paravirtualized and Emulated in power consumption level. The difference between Passthrough and SR-IOV for this parameter is not very considerable.

In the first compute node except the processing load and memory usage, other parameters affected the power consumption. However, also in this node the power consumption mostly follows the CPU usage, but analysing different metrics shows that the level of irq and type of processes (refer to table 5.1) were effective too. In this node Paravirtualized method with MTU 9500 was the most power consuming config. which its consumption was 10% more than SR-IOV (9500) and 16% more than Passthrough (9500).

Considering the amount of consumed power in both nodes during each experiment, Figure 5.9 show the the whole amount of energy (based on Watt-Hour) consumed by each method.

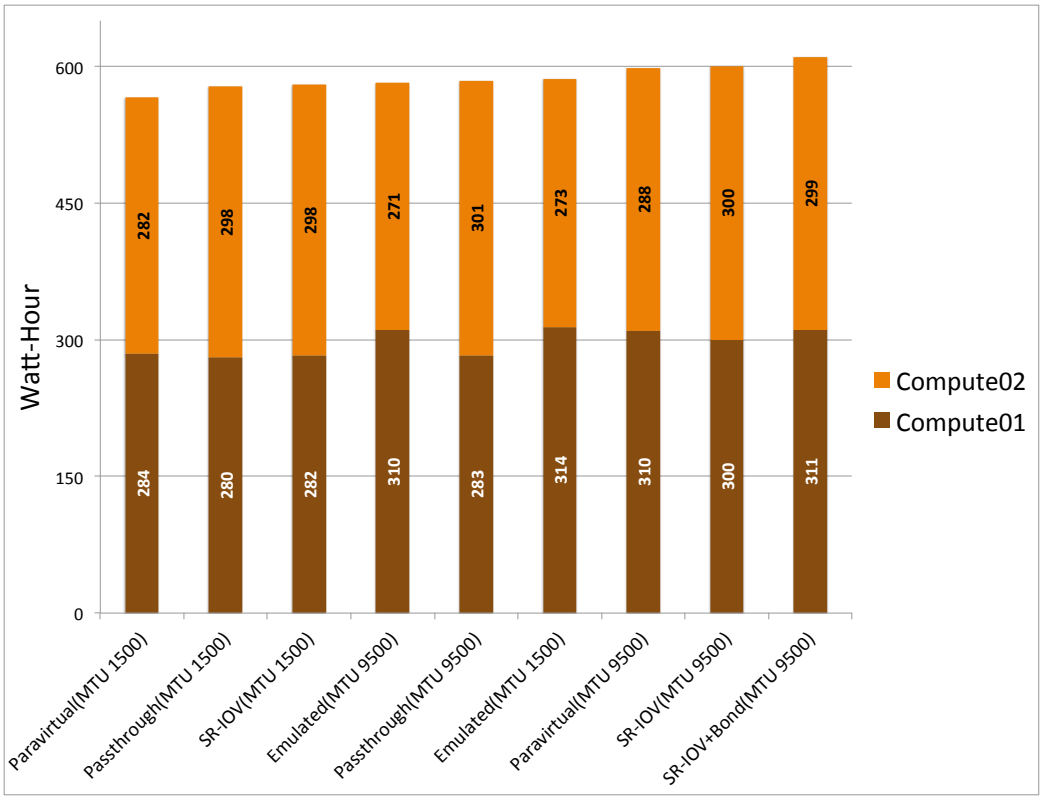


Figure 5.9: Energy consumption in whole system by experiments with single VM

Higher value in energy consumption does not necessarily mean lower efficiency. The element that should be considered is the amount of energy consumed to transfer the unit of data. Figure 5.10 shows the level of energy consumption per data ($\frac{Watt-hour}{GB}$) in both nodes. In another word it illustrates the total amount of energy that each of methods consumed to transfer 1 GB of data .

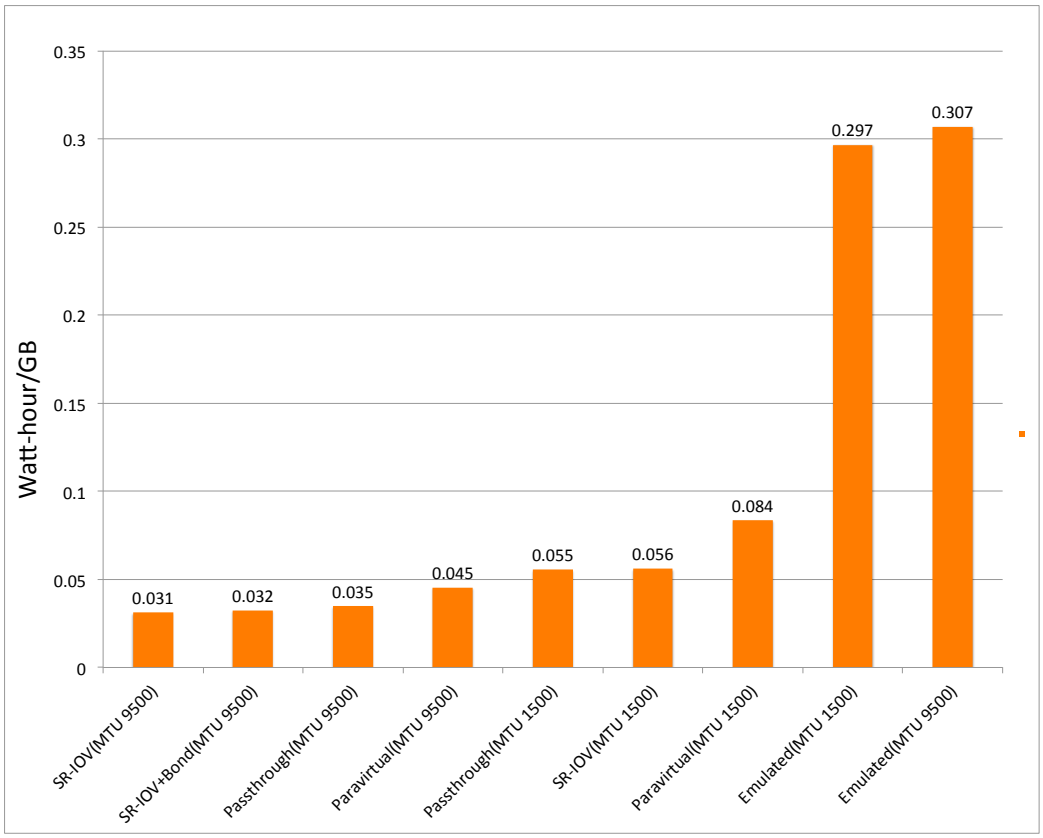


Figure 5.10: Energy consumption per data during experiments with single VM

In the first glance this numbers might seem too small and very close, but considering a huge amount of data and long time of working, this differences will be much more effective. For instance, to transfer 10000 GB of data the following amount of energy will be consumed:

Method	**one node**	**100 nodes**	
SR-IOV	310 WH	31000 WH	
Passthrough	350 WH	35000 WH	(+4000)
Paravirtual	450 WH	45000 WH	(+14000)
Emulation	3070 WH	307000 WH	(+276000)

This means in big scales (like middle size data centers) and with

continuous works, the amount of saved energy and cost reduction will be considerable.

5.1.2 Different methods with Multiple VM

In this part, obtained results from experiments with seven Virtual Machines are analyzed.

In terms of average bandwidth, outcomes from experiments with multiple VMs shows that SR-IOV and Paravirtualization delivered almost same throughputs. While they tried to utilize the bandwidth as much as possible, Emulation did not go far beyond its limitation. Figure 5.11 illustrates the average of delivered bandwidths by each method. These numbers are calculated by averaging the averages of 30 tests in each VM.

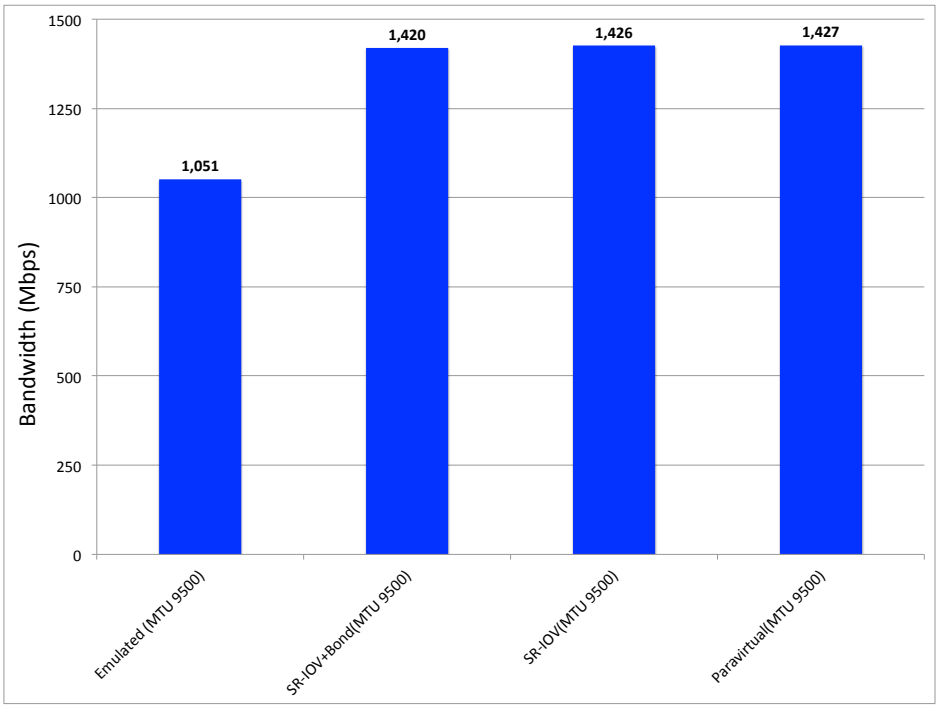
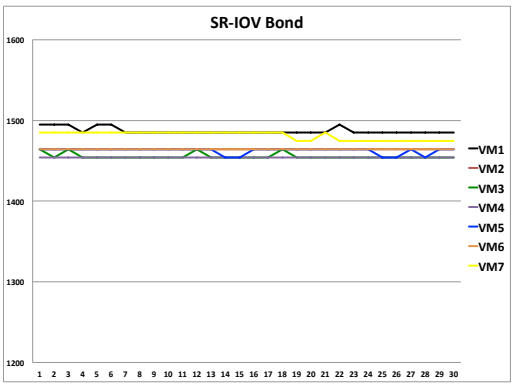


Figure 5.11: Average of average bandwidths in 7 VMs during experiments with multiple VMs

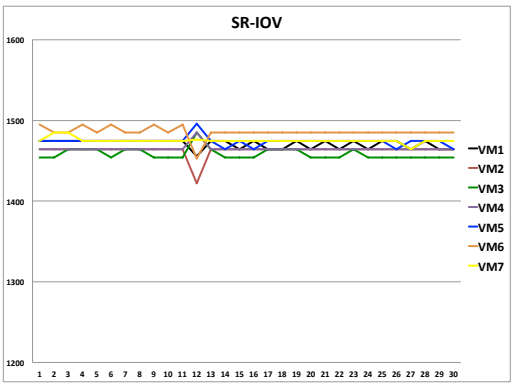
Concerning number of VMs, this graph shows that, except emulation other configurations utilized maximum possible bandwidth ($7 \times 1420 = 9940$ Mbps). The other understanding is that, even about Emulation, plurality in virtual environments (specifically clouds) could lead to better utilization of hardware. This means however a single VM with each of configurations did not meet the maximum throughput of NIC, but aggregation

of VMs performed it.

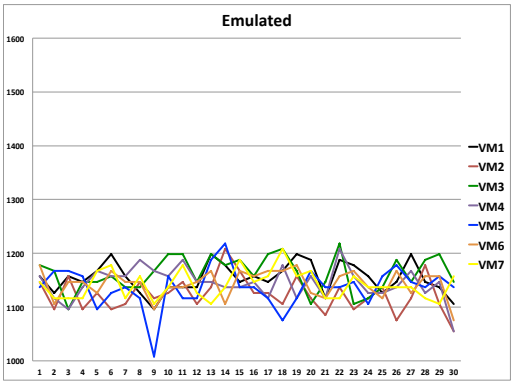
To have a better understanding about the quality of service by different methods, for each VM average bandwidth of all tests were plotted. These graphs help to observe the behavior of Network interface during each experiment. Figures 5.12a, 5.12b, 5.12c and 5.12d illustrate the average of bandwidth in all tests of all experiment .



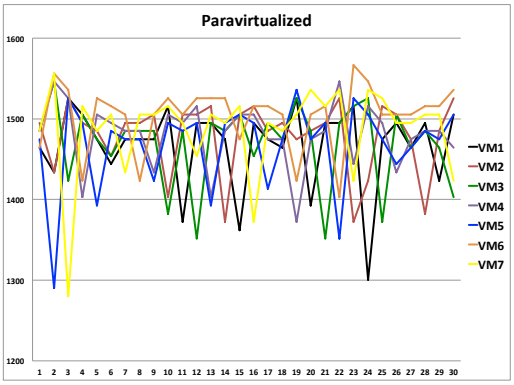
(a) Bandwidth average for all tests in all VMs during SR-IOV+Band experiment



(b) Bandwidth average for all tests in all VMs during SR-IOV experiment



(c) Bandwidth average for all tests in all VMs during Emulated experiment

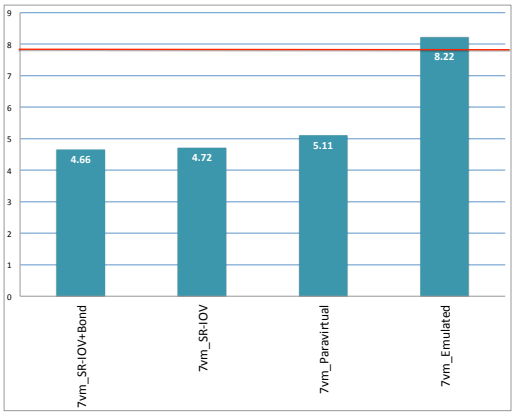


(d) Bandwidth average for all tests in all VMs during Paravirtualized experiment

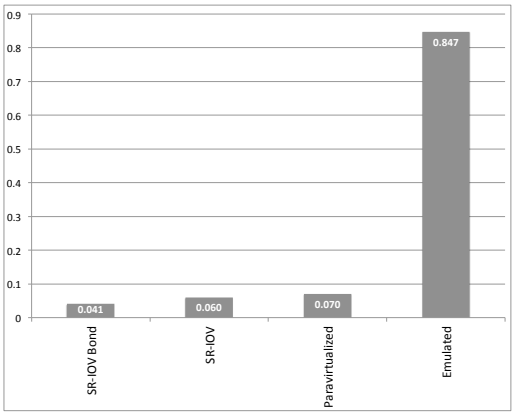
These plots clearly show that while both of SR-IOV configurations are perfectly stable and balanced, Paravirtualized and Emulated configurations are significantly unstable. It means that SR-IOV is greatly capable to maintain balanced and deliver equal service to all nodes. This matter would be more tangible while scaling the environment to a very large scale with hundreds of VMs. Also the quality of service in different layers will benefits from this stability.

Like previous experiments, impacts of increasing number of VMs on

different elements of system were analyzed. Figures 5.13a and 5.13a shows the load averages on Compute 01 and VMs so that for VMs the graph is average of all seven VMs during each experiment.



(a) Load average in Compute01 during experiments with 7 VMs



(b) Average of Load averages in VMs during experiments with 7 VMs

The load average inside the VMs for SR-IOV and Paravirtual is very low and for Emulated also is not critical, however it is much more than two others. SR-IOV has the lowest load and SR-IOV Bond also imposed lower load than Paravirtualized. Comparing to load average of VM during the previous experiments with single VM (refer to appendix C.3), numbers have been decreased. Taking the network bandwidth of each VM in consideration, this matter confirms the last finding that the load of the different systems linary changes based on network throughput.

But inside the host system load averages were increased almost considerably. While in case of single VM, Paravirtualized had the worst load average, this time the highest load average was presented by Emulation that exceeded the threshold. Compute node 1 was equipped by 8 cores so the threshold of the load should be considered by 8, and Emulation method passed this number . Since the overall bandwidth was enhanced by all configurations and number of VMs increased from 1 to 7, the load increase for other three configuration was acceptable. Comparing to single VM experiments, this increment was by 260% for Paravirtualized and 290% for both SR-IOV and SR-IOV Bond . It can be seen that still SR-IOV configurations remained better than Paravirtualized in terms of system load average. From above matter it can be obtained that based on system load, SR-IOV and Paravirtualization methods are much more scalable than Emulated and it would be a serious bottleneck for Emulation method.

Again the CPU usage information clearly describes the matter of load average. Figure 5.14 illustrates average of total CPU usage (including all elements) and the share of Interrupt request and Guest processes from this percentage.

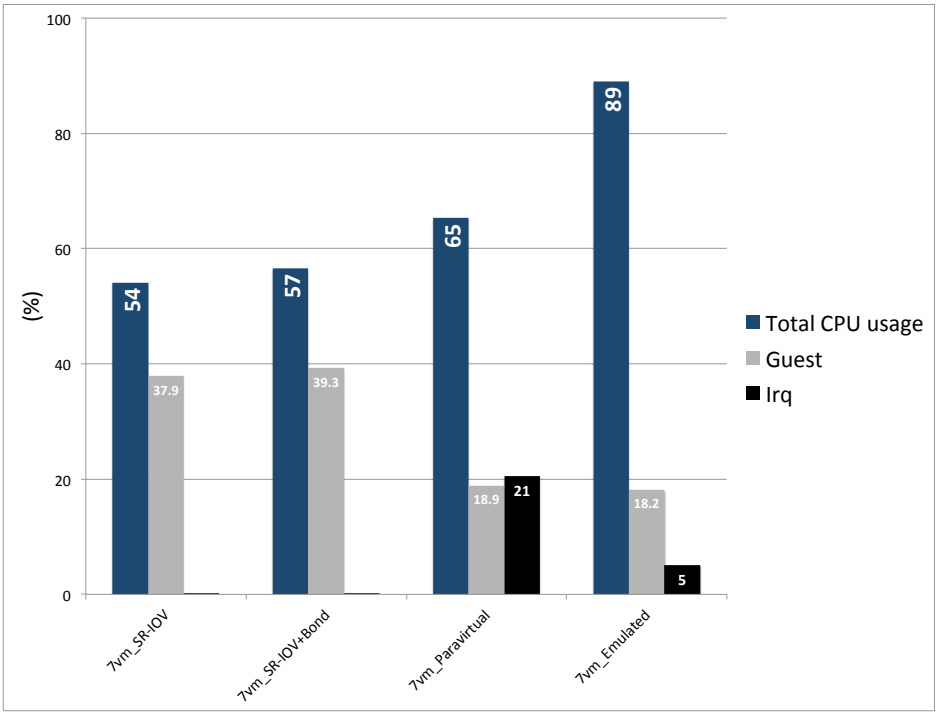


Figure 5.14: Average of CPU usage in Compute01 during experiments with 7 VMs

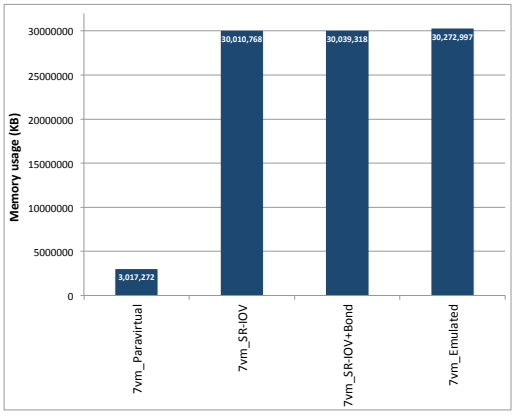
Comparing to outcomes from experiments with single VM, there is a significant change in CPU usage by Emulation. While in that case it had an equal usage to SR-IOV and much less than Paravirtualized, now its usage of CPU is 1.3 times more than Paravirtual and 1.6 times more than SR-IOV. The interrupt request for Emulation increased 25 times and the Guest processes that already was half of Paravirtualized, now is almost equal to it. But the most considerable point is the share of system processes which is about 65% of total CPU usage. Comparing to other configs, this increase of 375% in CPU usage means, Emulation seriously suffers from a huge load of kernel processes.

From one VM to 7 VMs, CPU usage of Paravirtualized increased by 136% in total while its IRQ increased by 114% and Guest processes by 170%. This increase for SR-IOV was by 184% in total and 196% in Guest that still makes it being better that Paravirtualized in CPU usage. Comparing the ratio of different elements in CPU usage for these two configur-

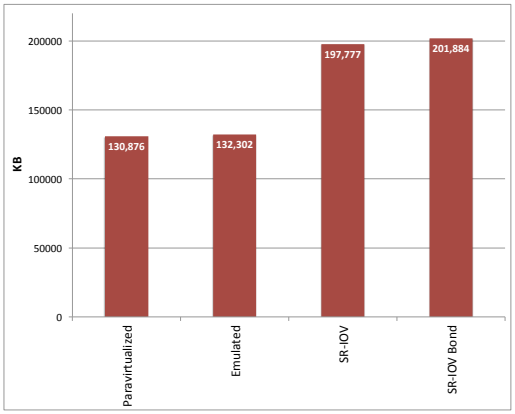
ation, and considering the scale of processing units inside host and VM, it can be concluded that the CPU cost of Paravirtualized in same situations is always more than SR-IOV. It means that utilizing the Paravirtualized device in itself imposes some overhead processes to system.

Inside the VM, CPU usage mostly followed the decrement of bandwidth (refer to appendix C.4), so that CPU usage for SR-IOV decreased by 58%, SR-IOV Bond by 65% and Paravirtual by 76%. But the CPU usage of Emulation remained the same as before. This confirms that a big part of CPU load inside the VMs during experiments are imposed due to generating a big amount of data and push it to network. Also from comparing processing loads of host and VM it can be concluded that against Emulation and Paravirtualization, SR-IOV method works independent and more isolated from VMM. Being isolated from host could bring security benefits as well. It is due to less interference by host to activities and traffics of VM.

Following graphs in figures 5.15a and 5.15b present the average of memory usage during experiments, inside the host and VM. This results show a very high level of memory usage by SR-IOV and Emulation in the host.



(a) Average of memory usage in Compute01 during experiments with 7 VMs



(b) Average of memory usage in VMs during experiments with 7 VMs

Inside the VM despite the reduced bandwidth, level of memory usage did not change substantially. There were a few increas for Paravirtualized and Emulated, and a few decrease for SR-IOV. It means that inside VMs memory usage mostly was in accordance to OS and applications activity and was not affected so much by bandwidth. In compute node 2 Memory usage for SR-IOV and Emulation increased equally and still they are very close (refer to appendix C.5). Also the memory usage for Paravirtualiza-

tion decreased to some extend In that node. Since the overall bandwidth did not change, this increase means memory usage of Compute2 was only affected by the change in number of senders (from 1 to 7).

In the host system (Compute1) memory usage for all configurations increased, so that Emulation had the biggest increase and Paravirtualization had the lowest. This increment was 20 times in Emulation, 6 times in SR-IOV and 2 times in Paravirtualization. Keeping almost same bandwidth (overly) by SR-IOV and Emulation , this matter was originated by increase in number of VMs.

All information regarding to memory usage from different nodes were analyzed again to find the root of this behaviour that resulted to such a high memory consumption. Results show that using SR-IOV and Emulation imposes a considerable memory overhead to the host system and it changes per number of VMs. Comparing the rate of memory usage with Paravirtualized and SR-IOV in different cases indicates that the origin of this overhead could be the SR-IOV driver (kernel module). For instance, in case of single VM with Paravirtualized NIC, only the host was using the *bnx2x* driver (Driver for the 10G interface) while with SR-IOV technique both host and VM were utilizing this driver separately. The memory usage inside Compute01 shows SR-IOV technique utilized the memory 2.7 times more than Paravirtualized. During experiments with multiple VMs, with almost same amount of data generation, inside the VMs SR-IOV used the memory 1.4 times more than Paravirtualized that this extra usage was the overhead of utilizing the driver. In that time the memory usage inside the host by SR-IOV was almost 10 times more than Paravirtualized and number of using the driver was 7 times more ($7 \times 1.4 = 9.8$). Since the whole content of guest memory is placed in user space of host memory, it can be concluded that SR-IOV driver imposes this extra overhead to host memory.

About the Emulation technique, referring to CPU usage informations, this very high memory usage is originated from considerably high kernel processes to emulate seven number of e1000 network interface.

Following figures 5.16 and 5.17 show the results of calculations on power consumption outputs. Analyzing those data showed that SR-IOV would be known as a really Green technology to improve networking.

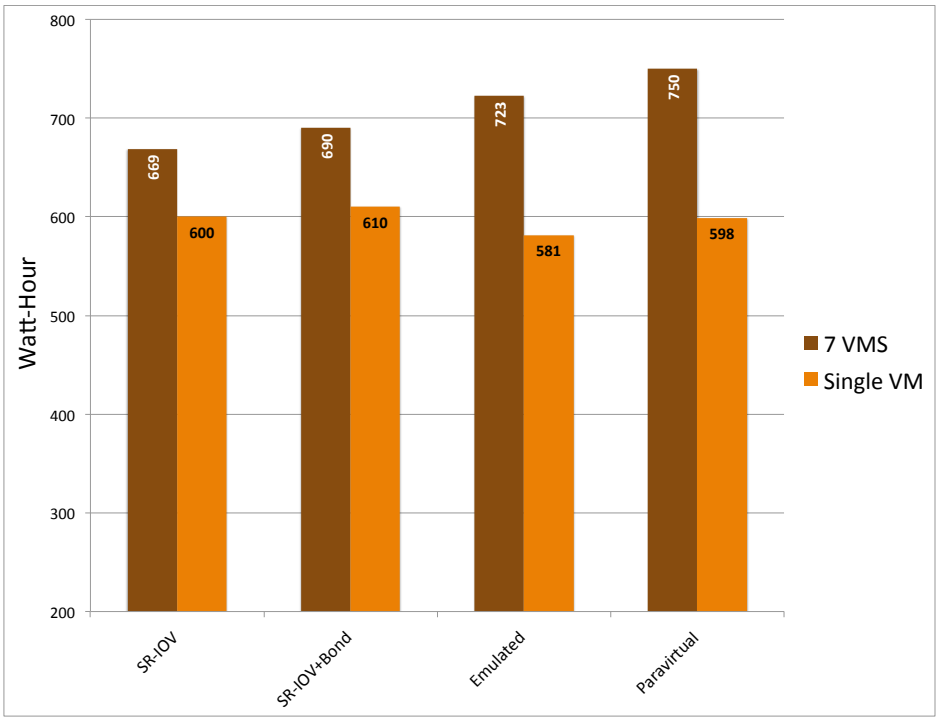


Figure 5.16: Total amount of consumed energy during all experiments in whole system

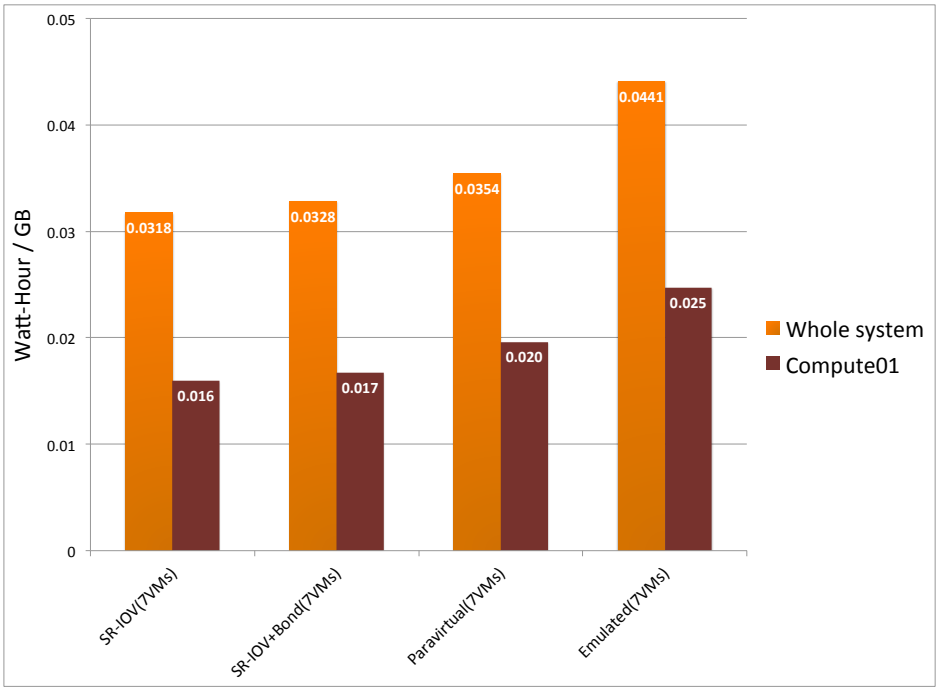


Figure 5.17: Rate of energy consumption per data during experiments in whole system and the host of VMs

It can be seen that however the amount of transferred data by SR-IOV(21086 GB) and Paravirtualized (21140 GB) was very close together

but the total amount of consumed energy by SR-IOV is less than Paravirtualized. This proves the previous calculations about the rate of energy consumption by each method for transferring data. Increasing the number of VMs led to increase of total energy consumption by 11% for SR-IOV, 13% for SR-IOV Bond, 24% for Paravirtualized and 25% for Emulated. It should be noticed that the overall bandwidth by all of them was increased so that SR-IOV and Paravirtualized reached the media limits.

Calculating the rate of energy consumption per data showed that still the cost of transferring data through SR-IOV is less than others. Since both SR-IOV and Paravirtualized reached the media limit, they were delivering equal bandwidth and transferred equal amount of data, but paravirtualized consumed more energy. This rate for SR-IOV in both case was equal and it was less than others. This matter indicates that energy consumption by SR-IOV changes linearly for different cases. It means from point of energy consumption, for any number of VM, its efficiency is stable while for few number of VMs the efficiency of Paravirtualized decreases.

5.2 Utilizing SR-IOV by OpenStack and Conducting Live Migration

As it described in the result section, the method of combining SR-IOV VF with Paravirtualized interface through Linux bonding driver, implemented and utilized for performing live migration. The implementation of method and performing the live migration were evaluated.

5.2.1 Evaluating Live Migration of SR-IOV attached VMs

All of experiments to test the live migration with SR-IOV were successful. By average the whole period of migration for non-busy VMs with only SR-IOV was 19.9 seconds and with bond interface was 20.9 seconds. These times for busy VMs was 23.2 and 25.6 seconds. This period was measured from time of pushing the button for running the migration command, to time of receiving traffic on SR-IOV interface at the destination. Following table 5.2 lists the details.

It should be mentioned that the time between detaching the device and calling the script (to reattach) at destination is included by a time which

Table 5.2: Times of different steps during Live Migration

		Bond	No Bond
Non-Busy	From command to call the script	2.1	2.1
	From call to perform Detach	1.16	1
	From detach to call script at destination	12.9	12.1
	From call to Reattach	2.89	1.7
	From Reattach to receive traffic	1.9	3
Busy	From command to call the script	1.8	2.1
	From call to perform Detach	2.8	1.2
	From detach to call script at destination	16.3	13.9
	From call to Reattach	2.8	2.8
	From Reattach to receive traffic	2.5	3.2

VM was in *Paused* mod. This period was in average 6.5 seconds for non-busy VMs and 9.1 seconds for busy VMs. The paused period was calculated based on recorded times inside the VM from the time gap during migration.

In order to make VMs busy, the *Stress* tool used to push load to CPU and make the memory busy. Without using bonding driver and slave interface, the down time of network was 17.69 seconds for non.busy and 21.6 seconds for the busy VM. The down time is the whole period between detaching the SR-IOV interface at origin, until receiving traffic on SR-IOV interface at destination.

Analyzing the recorded information from live migration experiments showed that the behaviour of the VM and networking configurations are according to assumption. Except the matter of incompatibility of linux kernel and SR-IOV driver (explained in discussion chapter 6.3.3) which led to a short network loss at compute 2, the rest of process was performed very well. Figures 5.18, 5.19, 5.20 and 5.21 illustrate the rate and amount of transferred data as well as delivered bandwidth to the VM1 with only SR-IOV interface. In all below plots the red lines indicate exact time of calling the sriov.sh script by nova, Green line indicates time of detaching VF, black line indicates the time that VM1 was paused at orging (acording to the gap in time stamps) and Blue line indicates the time of reattaching at destination.

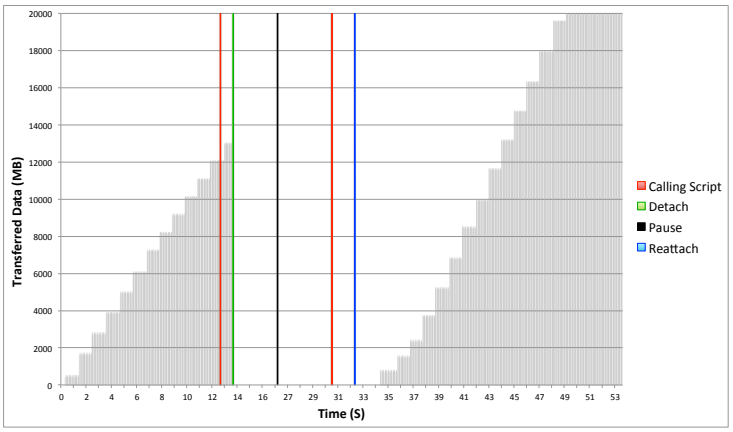


Figure 5.18: Amount and Rate of transferred data between VM1 and VM2 while migration from compute1 to compute2 - With only SR-IOV interface

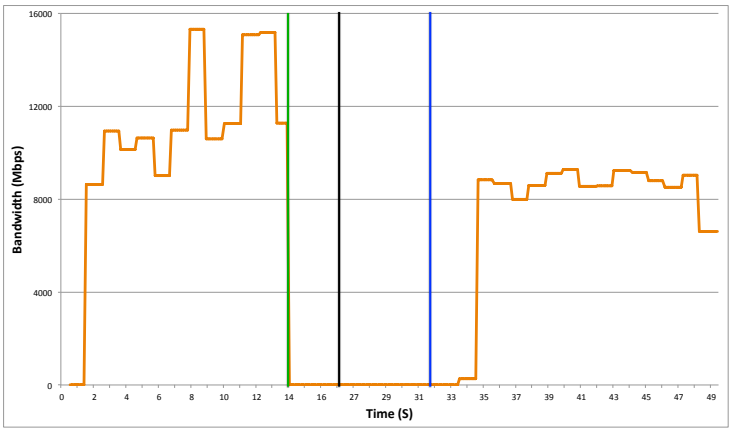


Figure 5.19: Provided bandwidth to VM1 while migration from compute1 to compute2 - With only SR-IOV interface

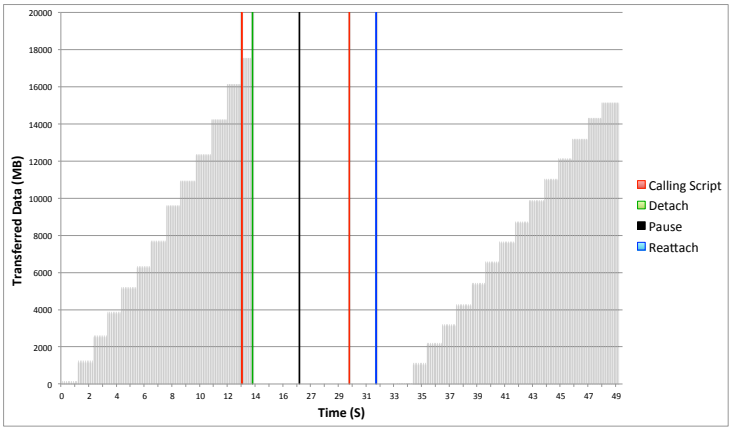


Figure 5.20: Amount and Rate of transferred data between VM1 and VM2 while migration from compute2 to compute1 - With only SR-IOV interface

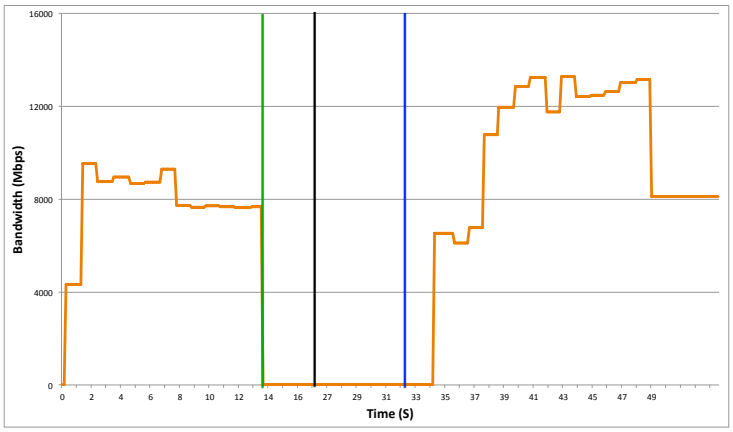


Figure 5.21: Provided bandwidth to VM1 while migration from compute2 to compute1 - With only SR-IOV interface

Above figures clearly show the real down-time of network while VM is equipped only with SR-IOV interface. This is the time between detaching the interface at origin host and retrieval of connectivity at destination. It can be seen that from the time of reattaching the interface to sending first packets, almost 2 seconds spend. This short period is due to operating system activities to reactive the interface. The reason that the counter of transferred data returned to 0 is that, by detaching interface from VM it will be disappear from /proc/net/dev and by attaching it gets new counter.

Figures 5.22, 5.23, 5.24 and 5.25 illustrate the rate and amount of transferred data and and the behavior of Bond interface during live migration of VM1 with bonding driver.

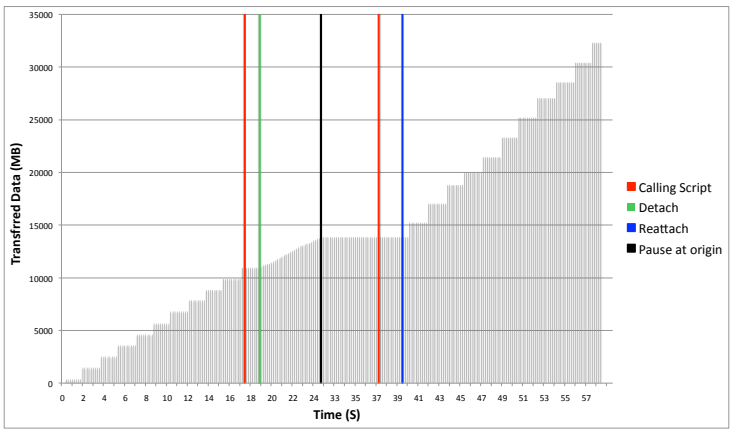


Figure 5.22: Amount and Rate of transferred data between VM1 and VM2 while migration from compute1 to compute2

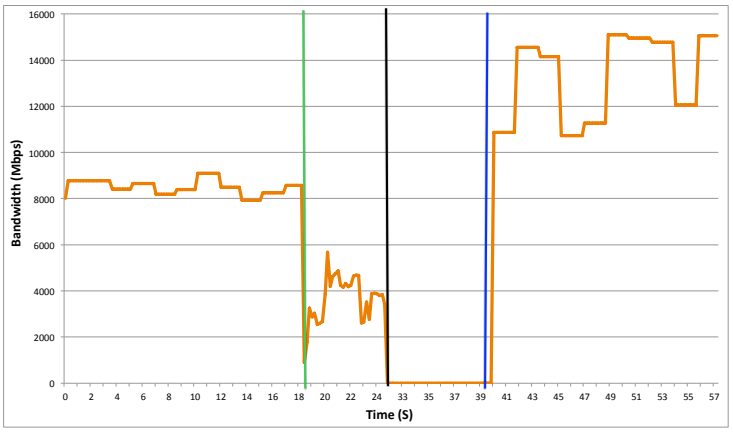


Figure 5.23: Provided bandwidth to VM1 while migration from compute1 to compute2

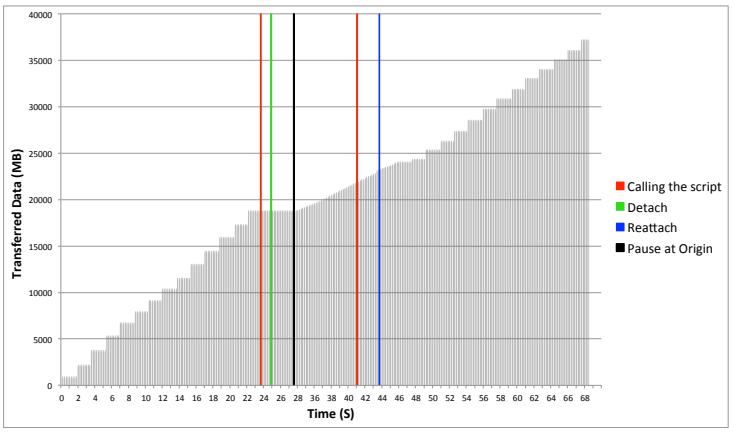


Figure 5.24: Amount and Rate of transferred data between VM1 and VM2 while migration from compute2 to compute1

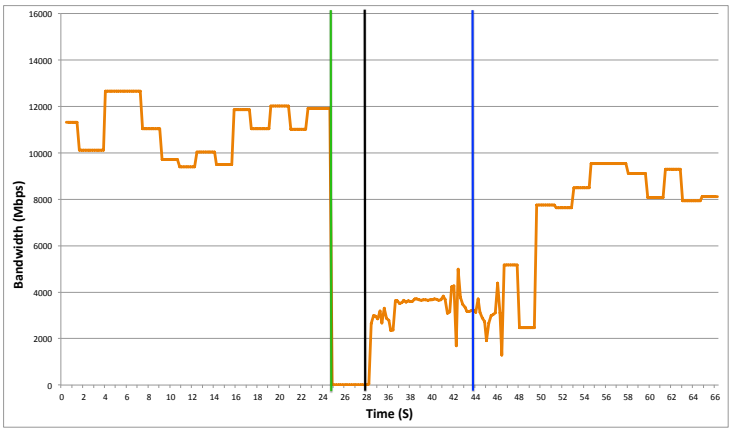


Figure 5.25: Provided bandwidth to VM1 while migration from compute2 to compute1

Figures 5.22 and 5.23 are related to migrating the VM1 from Compute1 to Compute2, where the VM2 is located. It can be seen that before live migration the slope of graph, which presents the rate of transferring data, is

almost 45 degree. About one second after calling the script, where SR-IOV was detached, the slope of graph is decreased and the pattern changed. Here is the period that the slave interface was transferring the data. This situation lasted for 6 seconds which is half of time between calls of script and at that point the VM1 was paused at origin. From this time the VM1 is activated in Compute 2 and due to issue of communication between VF and PF, no data was transferred for 7 seconds. It can be seen that from attaching the VF at destination until retrieval of connectivity at SR-IOV interface it took 0.5 second. After this point the slope of graph is even more than 45 degree which presents higher rate transmission. Further investigation on system showed that the only reason for stepwise increase in graph by SR-IOV is the interval of updating information for this interface and the big amount of transferred data during this interval .

Graph 5.23 shows the calculated bandwidth from above informations. This is the bandwidth that was provided to the VM1 during this experiment (it was seen was seen by bond0 interface). before starting the migration and detaching VF, the bandwidth is exactly according to previous experiences with SR-IOV . Immediately after detaching SR-IOV, paravirtualized interface started the transmission. From this point not only the bandwidth is decreased but also its pattern was changed that again is according to previous observations. Exactly at the middle of migration time VM1 arrived to Compute02 and lost the connectivity. Less than 1 second after reattaching again SR-IOV get the responsibility of networking. From this time, since both VMs are communicating on a same SR-IOV interface, the bandwidth exceeded the 10 Gbps and even in some points reached 15 Gbps.

Next two figures presents another tests of migration with same VM but this time from Compute2 to Compute1. In this experiment since at the start both VMs are on the same node, the slope of data graph is sharper at beginning. By detaching the VF the network connection was lost until 28th second (a little before half of migration time). At this time VM is activated at Compute1 and got back connectivity through paravirtualized NIC. About one and a half second after attaching VF in new host the SR-IOV took the control of networking. Following graphs 5.26 and 5.27 show the bandwidth of each interface during the migration experiment.

From all above graphs it can be concluded that by resolving the issue of

communication between VF and PF, the downtime of network will be decreased almost to 0 (Regardless from Paused period) . Moreover it can be seen that the period of retrieving connectivity on SR-IOV interface, while using bonding driver is a bit shorter. It might cause due to this fact that while using bonding driver, at the time of attaching SR-IOV, the networking is active.

From these graphs and previous observations from SR-IOV throughputs, it can be concluded that, by means of SR-IOV interface and migration technique it is possible to highly improve load balancing within cloud . The migration of VMs with SR-IOV is now possible and VMs in same node benefit from higher bandwidth while their traffic will not interfere to the whole network (communications between physical hosts).

Again it should be emphasized that by addressing the issue of kernel and driver through upgrading kernel and using proper driver/firmware, the network loss never happens and the VM keeps its connectivity during the whole period of migration.

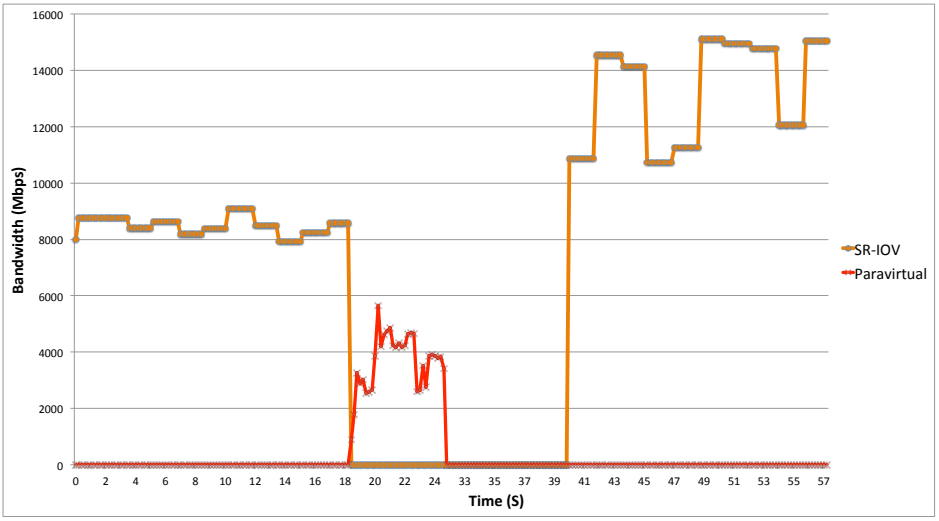


Figure 5.26: Provided bandwidth to VM1 while migration from compute1 to compute2 by each interface

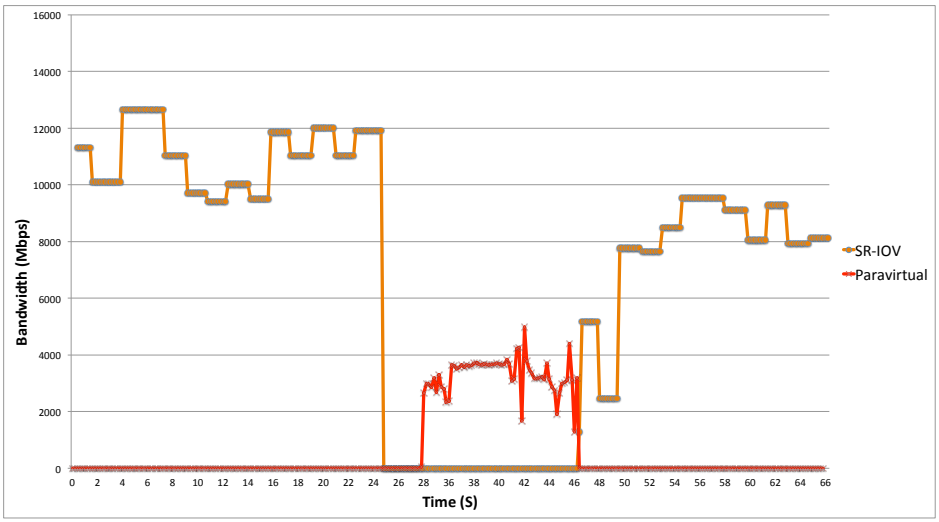


Figure 5.27: Provided bandwidth to VM1 while migration from compute2 to compute1 by each interface

Part III

Conclusion

Chapter 6

Discussion and Future works

This chapter discusses the different phases of this study, explain the technical difficulties during the project , and suggests improvements for future work and continuation of this study.

6.1 Evolution of the project as a whole

Performance analysis and dynamic reconfiguration of a SR-IOV enabled OpenStack cloud. The focus is mainly on utilizing SR-IOV technique for I/O virtualization in a cloud environment. This is to improve the networking performance and eventually the efficiency of IaaS. As it mentioned already in chapter 1, by deploying cloud environments achieving more efficiency is highly considered. Achieving efficiency is entirely dependent on improving the performance of system, and overall performance of the system is the result of performance of activities in subsystems. A cloud is a combination of different technologies and techniques distributed in different layers. Hierarchically higher layers are served by lower layers so that their functionality and performance depend on the lower layer. The lowest layer is IaaS which is the infrastructure of the cloud environment. IaaS is nothing but Networking and virtualization, and too many studies are ongoing to improve the performance of IaaS based on these two elements. SR-IOV is relatively new technology aimed to improve I/O virtualization and VM networking. By the literature survey of the project it was found that there are few studies around SR-IOV and its utilization. The efficiency of SR-IOV technique and impacts on networking performance, analyzed in this study.

Enabling OpenStack to use SR-IOV Virtual Functions leads to have

high-performance networking inside the VMs and reduce hypervisor intervention. It offers more stable networks and a Green functionality by efficient power consumption. Automating the process of assign and dissociate Virtual Functions enables live migration while using SR-IOV. It means this key feature of cloud will not be sacrificed for improving performance.

The biggest challenge throughout the accomplishment of the project was time constraints, and many technical difficulties. A part of initial plan was to demonstrate utilization of SR-IOV in OpenStack cloud for different cases, that did not conduct. Some configurations and changes were time consuming and needed more works so it was not possible to complete them during the time of this project. But since the OpenStack could utilize SR-IOV and do the live migration successfully, the other functions are also possible. Some future works can conduct them based on this study.

6.2 OpenStack deployment

To approach to the problem statement, this study first of all needed to deploy and configure an OpenStack testbed cloud computing. As it mentioned already due to benefit from the latest release and its features , HAVAN release was chosen. OpenStack official documents for HAVANA [1] was the only document availabe for this release and used as the guideline. These documents were being updated and debugging at that time .

This fundamental phase of project was more time consuming than expectation and ran out of timing plan. The deployment of OpenStack was not straightforward and several minor and few major problems were encountered. Asking questions in communities and getting replies or suggestions took a long time[2].

Configuration of OpenStack is totally based on too many configuration files for different services. This made it difficult to trace the root of errors even by reading several log files. In some configuration files the sequence of configuration codes was not important but in some of them it led to serious errors. For instance, in case of this project misplacing of just

[1] http://docs.OpenStack.org/havana/install-guide/install/apt/content/

[2] https://answers.launchpad.net/neutron/+question/243812 and https://ask.OpenStack.org/en/question/18577/error-400-while-trying-to-create-ext-net-gre/

two line of codes in a right file caused 4 days of tracing and troubleshooting. These lines was associated with messaging service (RabbitMQ) for neutron plug-in for Controller node. Since this mistake had led to malfunctioning network service, it seemed the root of problem should be in Network node and in main configuration files of neutron.

6.2.1 Neutron

Configuring the Neutron and make it to work was one of the challenging tasks. At first there was a challenge for troubleshooting its configuration since it was not working at all. After making it to work, it was found out that the Neutron does not provide access to external network for VMs. By more searches it became clear that in this specific version there was a bug for handling communication between internal and external networks with different subnet mask. The other Neutron bug that made some troubles for the project was providing IP for new VMs while creating group of instances[3]. Since some automation scripts were creating and configuring VMs for different experiments, this bug caused some problems and led to rerun some of experiments.

The initial plan was using neutron and doing configurations in a way to utilize SR-IOV with GRE or VLAN tag options. Due to other problems and lack of time this could not be reached. So the configurations changed to providing flat networking [4] with Neutron. Utilizing SR-IOV with complex networking needs to be done in future works.

6.3 Utilizing Ethernet SR-IOV enabled interface

6.3.1 The Issue of VF traffic

There are some famous providers for Ethernet SR-IOV enabled interfaces such as *Intel* and *Broadcom*, *Mellanox*, etc. Both compute nodes of this project were equipped by Broadcom interfaces. To enable and use Virtual Functions , it was needed to install the proper driver. Installing the available driver (gotten from providerÂ´s website) did not work since there

[3]https://bugzilla.redhat.com/show_bug.cgi?id=1023818

[4]http://developer.rackspace.com/blog/neutron-networking-simple-flat-network.html

was a problem with the interface. However the VFs could be assigned to VMs but there was no network traffic inside VMs. By some inquiries it was found out that this problem is caused by incompatibility between driver and kernel. Searching the issue ended up to finding Mr. Yoan Juet [5] from University of Nantes who had been working on a project with similar equipments. Based on his experiences with this specific hardware and some similar issue he pointed out the compatibility of driver, kernel and firmware.How ever he supported us with the driver, firmware and a customized kernel for this purpose, but the kernel did not work properly. As it was no new version of driver (and has not been released to this time) compatible with current kernel, both kernels on compute nodes were downgraded from 3.8.0 to 3.2.0. This solved the problem and made VMs to have access to the network.

6.3.2 The bug in libvirt

Another problem occurred with SR-IOV interface that took a considerable time to resolve. While testing the process of attach and detach VFs to/from VMs, it was found out that VFs are not detached properly from the VMs. Inquiries did not help but some tests showed that restarting the libvirt service after detach fixes the problem. So the problem reported on libvirt mailing list [6] and it turned out that this issue is a bug with *libvirt 1.1.1-0ubuntu8.5* and had not been reported until that time. In order to save the time immediately the libvirt was downgraded to *1.0.2-0ubuntu11.13.04*. The KVM and Libvirt were installed by OpenStack packages so downgrading only the libvirt needed adding Grizzly repositories, finding dependencies and pinning all of them to the version used in Grizzly release of OpenStack.

6.3.3 PF and VF communication issue

There was another technical difficulty with SR-IOV. affecting an important part of the project. While testing the proposed method for live migration VMs with SR-IOV, it was found out that there is no traffic between VF and PF (physical function) within an interface. It means VMs those equipped with SR-IOV VFs can communicate with each other or with other nodes

[5] https://www.univ-nantes.fr/juet-y/0/fiche___annuaireksup/&RH=INSTITUTIONNEL_EN
[6] https://www.redhat.com/archives/libvir-list/2014-March/msg01807.html

in the network, but they can not communicate with their host machine via SR-IOV interface. The following figure 6.1 illustrates this issue.

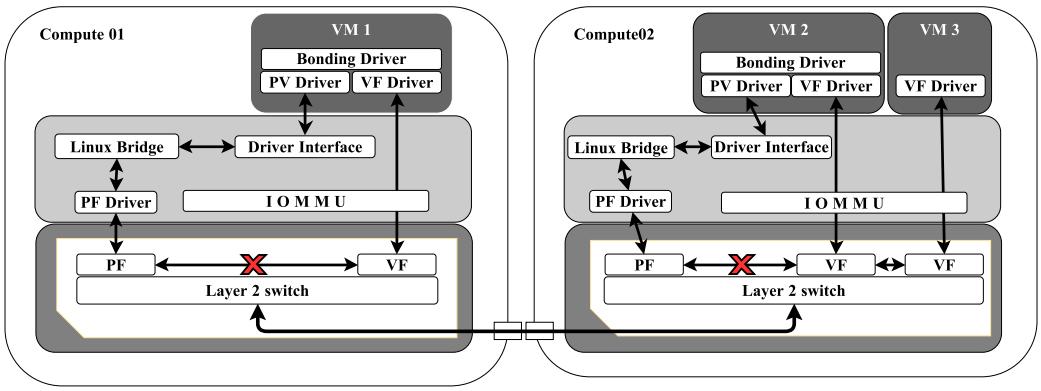


Figure 6.1: Communication between the various components and PF to VF communication issue

As it can be seen that the VM 2 and the VM3 can communicate with each other through VFs. Also via the network interface, they can communicate with Compute01 and VM1 through the network. But due to above problem they cannot communicate with Comput02. The same happens for VM1, while the SR-IOV interface is the active slave for bonding driver. It can not communicate to its host (Compute01) but it can communicate with other nodes. If the VF interface become disabled during migration, it will establish a connection to Compute1 through the Paravirtualized interface and linux bridge. When it is moved to Compute2, at arrival it will be connected to the bridge through its PV interface. Before attaching the new VF at Compute2, it can communicate with the host but it can not communicate with VMs.

This issue had taken place again due to matter of Driver and Kernel. The same issue for intel interfaces had been debugged by intel developper team and was working in newer version of kernel (3.8.5) [7]. Hoewer upgrading kernel to 3.8.5 might cause some difficulties, but In order to properly perform this phase of the project the kernel upgraded. This move did not solve the problem since the driver was not updated by the provider. The decision was to continue project with this issue and show the ability of using SR-IOV and migration with current configuration. This issue does not overshadow the project and its purpose since clearly the

[7] https://communities.intel.com/thread/38613

migration was successful. It can be completely solved in future works with updated driver and kernel.

6.3.4 Security Concerns about SR-IOV

However it was not aimed in this project to work on security threats of SR-IOV, but during studies few security concerns drew attention. Some people reported a security concerns about traffic isolation [8] between PF and VF or VFs with each other . It seems there is no issue or was solved by intel in its products. No document or report was founded about solving this issue by Broadcom.

6.4 Changes in initial plan

Except the change in initial idea of using Neutron complex networks which happened due to technical issues and lack of time, another change took place during the project. The initial plan for investigational experiments was to use the maximum possible VF during tests. It aimed to evaluate the capabilities of SR-IOV interface and demonstrating its scalability. The expectation was 64 VFs per interface but in case of this project with current firmware, driver and kernel, only 8 VFs were available in each interface, so the scripts designed to create 8 VMs and attach all 8 VFs. By initiating the first experiment with multiple VMs, the host (Compute1) first get very slow and after a while crashed. Reviewing the log files did not provide a clear answer since all records showed normal behaviour. More investigations indicated that attaching the 8th VF caused the failure. However no similar case was found but reviewing different documents and reports showed that such a case can be caused by kernel. It should be mentioned that before downgrading the kernel on the same host 16 VFs were available per interface. This experience proved that the this issue is caused by kerne as well. using 7 VFs was found safe and reliable for rest of project.

All above mentioned issues show that SR-IOV is relatively a brand new technology and still needs some more works in different aspects. To use SR-IOV in an operational environment updating and coordinating all components and applications should be considered.

[8] https://www.mail-archive.com/libvirt-users@redhat.com/msg05837.html

6.5 Future works

This study was an initial work to investigate the functionality of SR-IOV and utilize it in a cloud environment. Lack of equipments and time constraints did not allow for further and more accurate studies. Many more things could be considered and conducted during the study and running the project. Thus a part of idea for this study and some other issues need to be examined and evaluated in future works.

1. Some outcome and results of experiments in this project were difficult to interpret and analyze. For instance the very high level of memory usage during the experiments with multiple VMs was hard to interpret and find the exact cause. Also the the precise relationship between power consumption and different activities inside the system was difficult to find. To measuring the real consumed energy by each configuration some other effective parameters to the power like temperature and fan speed could be collected. Considering more parameters and collecting more data from different indicators could help the more accurate analysis and investigation about all impacts and consequences of utilizing SR-IOV and different methods. Moreover an isolated and more standard test environment could guarantee the results of measurements and evaluation.

2. Equipments of this study and the issue of time did not allow to perform more experiments with SR-IOV in more real situations. Highly active cloud would have a busy network due to traffics of standard procedures of cloud (like updating information that are broadcast frequently). This matter would affect on real throughput of network. Considering this issue, more experiments could be conducted to measure the the real throughputs of other methods.

3. The initial idea of this study aimed to demonstrate the functionality of SR-IOV in the cloud while performing live migration based on some scenarios of use cases. This could be conducted to present usability of SR-IOV and demonstrating successful functionality of the method. Since the current study successfully conducted the live-migration of VM with SR-IOV interface, so in any case of need it is possible to migrate VMs. During the project, based on monitoring different indicators of the system, some results about behaviour of configurations were obtained and some relations were discovered.

These informations can be used to keep the system in a balanced condition or predict the next state of system. A script can be written using these information and calculations as the knowledge base and utilizing some of the tools used for this project (e.g. ipmitool) to dynamically reconfigure the environment. For instance based on level of system indicators, it is possible to evaluate the current QoS of the system (according to standards or local definitions) and compare it to the thresholds. At any time if it exceed the threshold, scripts decide to migrate some VMs for adjustment. The other scenario was to evacuate one host due to maintenance or fault tolerance. Using ipmitool many sensors such as power supplies and fans speed can be monitored. Missing one of power supplies or similar cases should raise a red alarm which requires a prompt reaction. The script can evacuate the malfunctioning host and turn it off. Based on what stated about energy efficiency of SR-IOV in analysis chapter, above function also can be used for energy saving purposes.

4. Study on usage of SR-IOV with complex virtual networks (e.g. GRE tunneling) and effects of those configurations on SR-IOV functionality and throughputs.

5. Based on different observations during the project and a comparison between results of different experiments with Single VM and multiple VMs, It is conjectured that Paravirtualized interface (as the only competitor of SR-IOV) might not exceed the limitation of 6 to 7 Gbps in one node (in case of single VM or very few VMs). Even utilizing more powerful and capable hardwares might not considerably improve this limitation. There are very high speed interfaces (i.e. 40Gb Ethernet interfaces) which offer bandwidth even higher than 10Gbps. A study similar to this project (experiments) with this equipments could present the unique performance and ability of SR-IOV.

Chapter 7

Conclusion

The problem statement for this thesis was : To provide improved network performance and dynamic reconfiguration of underlying infrastructures, for OpenStack IaaS cloud providers using SR-IOV capable Ethernet cards. This must be done in a transparent way to the end user. To understand features and capabilities of SR-IOV, this technique was compared to other techniques of I/O virtualization. All of four available methods to provide virtual network interface for Virtual Machines were implemented in an OpenStack cloud environment. During similar experiments their functionality were tested and evaluated from different aspects. By increasing the number of Virtual Machines their scalability and impacts on environment were examined. By writing some scripts and editing OpenStack codes, this IaaS provider was enabled to use the SR-IOV VFs for Virtual Machines automatically at the time of building. Additionally using same scripts, OpenStack was configured to run the live migration of VMs in transparent way to the user. Implementing the technique of "combining SR-IOV and Paravirtualized interfaces through Linux Bonding Driver" inside the VM , almost eliminated the down time of network during the Live Migration. All of the initial questions have been answered but while running the project some questions emerged which are suggested for the future works. Following list is the answers to corresponding research questions of section 1.1 :

1. In terms of bandwidth for normal configuration (Normal frame size) on a 10Gb interface, Passthrough and SR-IOV delivers an equal throughput which is about 4,900 Mbps. This throughput is 1.5 times (50%) higher than Paravirtualized (3,200 Mbps) and 5.4 times (440%) higher than Emulated (914 Mbps). By increasing the frame size for maximum utilization of bandwidth, SR-IOV delivers a considerably

high bandwidth (in average 9133 Mbps) . In this case its throughput is 1.15 times higher than Passthrough (7,900 Mbps), 1.5 times higher than Paravirtualized (6200 Mbps) and 10 times higher than Emulated. Using SR-IOV not only improves the network bandwidth but also makes it more stable comparing to Paravirtualized and Emulated. This stability leads to delivering equal service to all users in higher layers and consequently improving the quality of provided service (QoS) by IaaS provider.

In terms of system load and CPU consumption, with the standard frame size, again Passthrough and SR-IOV have similar behaviour and their imposed to the system is 1.16 times less than both Paravirtualized and Emulated. By increasing the frame size Paravirtualized method has considerably higher load than other methods. With 47% lower bandwidth than SR-IOV, Paravirtualized imposes 19% more load to the system. Increasing frame size also makes SR-IOV to have a load by 9% more than Emulated but considering its 900% higher bandwidth, it would be very efficient.

Increasing the number of VMs showed that however Passthrough delivers a good throughput, but forasmuch as scalability issue, it wouldn't be useful for cloud environments. In case of multiple Virtual Machines, SR-IOV offers a perfectly balanced and stabled networking while Paravirtualized and Emulated can not deliver same quality of service. It became clear that Emulated can not exceed its limit of 1 Gbps in any cases. Although Paravirtualized and SR-IOV provide equal bandwidths (in average) for each VM in case of multiple VMs, the results showed that SR-IOV performs more efficient that Paravirtualized.

In comparison with Paravirtualized, the memory consumption of SR-IOV is a drawback. This issue needs more detailed study and more standard tests to be clarified. Based on current results of this study, memory usage of SR-IOV is the only disadvantage comparing to Paravirtualized

2. In case of single VM, SR-IOV configuration consumes a bit more energy than others, but considering its throughput, SR-IOV has the lowest energy consumption for transferring data (Energy per unit

of Data). By increasing the number of VMs SR-IOV has the lowest increase in energy consumption. It clearly indicates that SR-IOV is the most efficient method and in large scales it would deliver highly efficient work. Consequently SR-IOV would be known as a Green technology in computing area.

3. The prototype solution that presented in this project for enabling OpenStack to use SR-IOV Virtual Functions for its VMs, has two steps. The first is to create one or couple of script(s) to automate process of attaching VF to VM and the second is editing OpenStack codes (Specifically drive.py) to use automation scripts at the time of creating VM. The process of attaching a VF to VM includes generation a MAC address for VF, creating a XML file (based on libvirt format) and using QEMU (through libvirt) for attaching function. Instead of creating additional scripts, this process could be appended directly to OpenStack functions by adding lines of codes to OpenStack standard scripts. This prototype solution was sufficient to demonstrate what are needed for this purpose.

4. Based on observations in this study, using only the SR-IOV device, the down time of network during live migration is: the whole time between detaching the device at origin, and a bit after reattach it at destination. After attaching device at destination, it takes a short while to to retrieve the connectivity. According to the results of this study, for non busy VMs (with specifications of VM in this project) the down time of network (in average) was 16800 milliseconds and for busy VMs 19900 milliseconds. This was tested in same VMs.

5. In order to enable live-migration of VM with SR-IOV, the OpenStack should be configured by editing its codes (drive.py and manager.py) to use supportive scripts to detach the VF at origin and attach it at destination. To have the same interface at destination from VM point of view, it is needed to attach the VF with same MAC address. To make this process transparent-to-user, the down time of network should be reduced or even eliminated so that user never feels the migration. To reduce this time, the method used in this project is to replace the SR-IOV interface with Paravirtualized interface during the migration and replacing it again with SR-IOV promptly at destination. To handle this replacement process Linux Bonding Driver is used to aggregate SR-IOV and Paravirtualized interface

inside the VM. To make it more transparent-to-user, customized cloud images can be used for creating VMs with some initial configurations (e.g. bonding driver) so that no intervention to the VM is needed after its creation.

According to this study, utilizing SR-IOV technique not only increases the performance of networking, but also does not sacrifice the the ability of dynamic reconfiguration within cloud. However, the technique used in this study is not completely transparent to the tenants since they need to have the bonding driver installed in the VMs, or use preconfigured images. More research and development needs to be done to enable SR-IOV in public clouds and keep the dynamic reconfiguration benefits provided by live-migration.

Appendices

Appendix A

System setup and configuration

A.1 Nova

Nova configuration file: /etc/nova/nova.conf

```
[DEFAULT]
auth_strategy=keystone
dhcpbridge_flagfile=/etc/nova/nova.conf
dhcpbridge=/usr/bin/nova-dhcpbridge
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova
force_dhcp_release=True
iscsi_helper=tgtadm
libvirt_use_virtio_for_bridges=True
connection_type=libvirt
root_helper=sudo nova-rootwrap /etc/nova/rootwrap.conf
verbose=True
ec2_private_dns_show_ip=True
api_paste_config=/etc/nova/api-paste.ini
volumes_path=/var/lib/nova/volumes
enabled_apis=ec2,osapi_compute,metadata
rpc_backend = nova.rpc.impl_kombu
rabbit_host = controller
rabbit_password = *****
my_ip=192.168.10.1

vncserver_listen=192.168.200.231
vncserver_proxyclient_address=192.168.200.231

neutron_metadata_proxy_shared_secret = *****
service_neutron_metadata_proxy = true
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://controller:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=*****
neutron_admin_auth_url=http://controller:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver=nova.virt.firewall.NoopFirewallDriver
security_group_api=neutron

[database]
connection = mysql://nova:NOVAPASSWORD@controller/nova

[keystone_authtoken]
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = ****

[novncproxy]
novncproxy_host=0.0.0.0
```

```
novncproxy_port=6080
novncproxy_base_url=http://192.168.200.231:6080/vnc_auto.html
vncserver_listen=192.168.200.231
vncserver_proxyclient_address=192.168.200.231
vnc_enabled=true
```

Nova configuration file: /etc/nova/api-paste.ini

```
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host = controller
auth_port = 35357
auth_uri = http://controller:5000/v2.0
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = *****
# signing_dir is configurable, but the default behavior of the authtoken
# middleware should be sufficient.  It will create a temporary directory
# in the home directory for the user the nova process is running as.
#signing_dir = /var/lib/nova/keystone-signing
# Workaround for https://bugs.launchpad.net/nova/+bug/1154809
auth_version = v2.0
```

A.2 Neutron

Neutron configuration file: /etc/neutron/neutron.conf

```
[DEFAULT]
auth_strategy = keystone
state_path = /var/lib/neutron
lock_path = $state_path/lock

core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.\
OVSNeutronPluginV2

allow_overlapping_ips = False

rabbit_host = controller
rabbit_userid = guest
rabbit_password = ****

notification_driver = neutron.openstack.common.notifier.rpc_notifier

[agent]
root_helper = sudo /usr/bin/neutron-rootwrap /etc/neutron/rootwrap.conf

[keystone_authtoken]
auth_host = controller
auth_port = 35357
auth_protocol = http
```

```
admin_tenant_name = service
admin_user = neutron
admin_password = ****
signing_dir = $state_path/keystone-signing

[database]
connection = mysql://neutron:NEUTRONPASSWORD@controller/neutron


[service_providers]
service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.\
drivers.haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

Neutron configuration file: /etc/neutron/api-paste.ini

```
[composite:neutron]
use = egg:Paste#urlmap
/: neutronversions
/v2.0: neutronapi_v2_0

[composite:neutronapi_v2_0]
use = call:neutron.auth:pipeline_factory
noauth = extensions neutronapiapp_v2_0
keystone = authtoken keystonecontext extensions neutronapiapp_v2_0

[filter:keystonecontext]
paste.filter_factory = neutron.auth:NeutronKeystoneContext.factory

[filter:authtoken]
paste.filter_factory= keystoneclient.middleware.auth_token:filter_factory
auth_host = controller
auth_uri = http://controller:5000
admin_tenant_name = service
admin_user = neutron
admin_password = NetPass01


[filter:extensions]
paste.filter_factory=neutron.api.extensions:plugin_aware_extension_\
middleware_factory

[app:neutronversions]
paste.app_factory = neutron.api.versions:Versions.factory

[app:neutronapiapp_v2_0]
paste.app_factory = neutron.api.v2.router:APIRouter.factory
```

A.3 Nova-Compute

Compute node configuration file:/etc/nova/nova.conf

```
[DEFAULT]
auth_strategy=keystone
dhcpbridge_flagfile=/etc/nova/nova.conf
dhcpbridge=/usr/bin/nova-dhcpbridge
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova
force_dhcp_release=True
iscsi_helper=tgtadm
libvirt_use_virtio_for_bridges=True
connection_type=libvirt
root_helper=sudo nova-rootwrap /etc/nova/rootwrap.conf
verbose=True
ec2_private_dns_show_ip=True
api_paste_config=/etc/nova/api-paste.ini
volumes_path=/var/lib/nova/volumes
enabled_apis=ec2,osapi_compute,metadata

rpc_backend = nova.rpc.impl_kombu
rabbit_host = controller
rabbit_password = RabbitPass01


my_ip=192.168.10.3
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=192.168.10.3
novncproxy_base_url=http://controller:6080/vnc_auto.html

glance_host=controller

network_api_class=nova.network.neutronv2.api.API
neutron_url=http://controller:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=NetPass01
neutron_admin_auth_url=http://controller:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver=nova.virt.firewall.NoopFirewallDriver
security_group_api=neutron


[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://nova:NovaPass01@controller/nova
```

Compute node configuration file:/etc/neutron/neutron.conf

```
[DEFAULT]
state_path = /var/lib/neutron
lock_path = $state_path/lock

core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.\
OVSNeutronPluginV2

rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_host = 192.168.10.1
rabbit_port = 5672
rabbit_password = ****

notification_driver = neutron.openstack.common.notifier.rpc_notifier

[agent]
root_helper = sudo /usr/bin/neutron-rootwrap /etc/neutron/rootwrap.conf


[keystone_authtoken]
auth_uri = http://controller:5000
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = neutron
admin_password = ****
auth_url = http://controller:35357/v2.0
auth_strategy = keystone
signing_dir = $state_path/keystone-signing

[database]
connection = mysql://neutron:NEUTRONPASSWORD@controller/neutron

[service_providers]
service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.\
drivers.haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

A.3.1 Reconfiguring QEMU and Libvirt for Live Migration

Script to edit QEMU and Libvirt

```
apt-get install -y kvm libvirt-bin pm-utils

# Edit /etc/libvirt/qemu.conf
sed -i 's/#cgroup_device_acl = \[/cgroup_device_acl = \[/' /etc/libvirt/qemu.conf
sed -i 's/#"\/dev\/null","\/dev\/full","\/dev\/zero",/"\/dev\/null","\/dev\/full",\
"\/dev\/zero",/' /etc/libvirt/qemu.conf
sed -i 's/#"\/dev\/random", "\/dev\/urandom",/    "\/dev\/random", "\/dev\/urandom",/'\
/etc/libvirt/qemu.conf
sed -i 's/#"\/dev\/ptmx", "\/dev\/kvm", "\/dev\/kqemu",/  "\/dev\/ptmx", "\/dev\/kvm",\
"\/dev\/kqemu",/' /etc/libvirt/qemu.conf
sed -i 's/#  "\/dev\/rtc","\/dev\/hpet"/ "\/dev\/rtc","\/dev\/hpet", "\/dev\/net\/tun"/'\
/etc/libvirt/qemu.conf
sed -i 's/#\]/\]/' /etc/libvirt/qemu.conf
sed -i 's/^#security_driver.*/security_driver = "none"/' /etc/libvirt/qemu.conf

# Edit /etc/libvirt/libvirtd.conf
sed -i 's/#listen_tls = 0/listen_tls = 0/' /etc/libvirt/libvirtd.conf
sed -i 's/#listen_tcp = 1/listen_tcp = 1/' /etc/libvirt/libvirtd.conf
sed -i 's/#auth_tcp ="sasl"/#auth_tcp ="sasl"\nauth_tcp ="none"/'/etc/libvirt/libvirtd.conf

# Edit /etc/init/libvirt-bin.conf
sed -i 's/env libvirtd_opts="-d"/env libvirtd_opts="-d -l"/' /etc/init/libvirt-bin.conf

# Edit /etc/default/libvirt-bin
sed -i 's/libvirtd_opts="-d"/libvirtd_opts="-d -l"/' /etc/default/libvirt-bin

virsh net-destroy default
virsh net-undefine default

service dbus restart && service libvirt-bin restart
```

Appendix B

Developped Scripts

B.1 Experiment tools

B.1.1 Load.pl

Script Load.pl

```
#!/usr/bin/perl

# needed packages
use strict "vars";
use Getopt::Std;
use Time::HiRes qw(gettimeofday);

# Global Variables
my $VERBOSE = 0;
my $DEBUG = 0;

# commandline options
my $opt_string = "vdhs";
getopts( "$opt_string", \my %opt ) or usage() and exit(1);

$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};
# my $server=$opt{'a'};
# print "$server\n";

if ( $opt{'h'} ) {
    usage();
    exit 0;
}
if ( $opt{'s'} ) {
        stop();
        exit 0;
    }


## Main part ##########################################################

my $row =0;

my $MemFile="/proc/meminfo";
my $CPUFile="/proc/stat";

my $freememory;
my $totalmemory;

my $buffer;
my $SwapTotal;
my $SwapFree;
my $cached;

my $systemtime;

my @cpu;
my @Ncpu;
my $CPUcount;

my $NICcount;
my @NICname;
my @NICdata;

my $load;

my $hostname=qx(hostname);
chomp($hostname);

my $nametime=qx(echo \$(date +\%s\%M));
```

```
chomp $nametime;

my $outfile="Load_Record_" . $hostname . "_". $nametime . ".csv";

my $checkfile="check.temp";

print "\n\nRecording is started\n";
print "\n The out put will be recorded in ($outfile) file located in current directory\n";
print "\n The check file is  ( $checkfile ) located in current directory\n";

qx (touch $checkfile | echo "1" > $checkfile);
my $check=qx(cat $checkfile);

my $counter=0;

########################### Counting Number of CPUs ###############################

open (CPU, '/proc/stat') or die "Error opening file  $!\n";

while (my $line = <CPU>) {

        if ($line =~ /cpu\d\s(.*)/ ){
            $CPUcount++;
        }
}
close CPU;
############################## Counting Number of NICs ##############################
open (NIC, '/proc/net/dev') or die "Error opening file  $!\n";

while (my $line = <NIC>) {

    if ($line =~ /([a-zA-Z]+.*\d*):\s*(.*)/ ){

        $NICcount++;
        push (@NICname,$1);
    }
}
close NIC;
############## Creating Output File and Writing Header of file ######################

open  (FILE, ">>$outfile") or die "Error creating file $!\n";

print FILE "Raw,Month,Day,Time,TotalMemory,FreeMemory,Buffers,Cached,SwapTotal,SwapFree," ;
print FILE "TotalUser,TotalNice,TotalSystem,TotalIdle,TotalIOwait,Totalirq,TotalSoftirq,\
TotalSteal,TotalGuest,TotalGuestNice," ;

for (my $i=0; $i < $CPUcount; $i++) {
    print FILE "CPU$i-User,CPU$i-Nice,CPU$i-System,CPU$i-Idle,CPU$i-IOwait,CPU$i-lirq,";
    print FILE "CPU$i-Softirq,CPU$i-Steal,CPU$i-Guest,CPU$i-GuestNice,"
}

for (my $i=0; $i < $NICcount; $i++) {
    print FILE "$NICname[$i]RCVbytes,$NICname[$i]RCVpackets,$NICname[$i]RCVerrs,";
    print FILE "$NICname[$i]RCVdrop,$NICname[$i]RCVfifo,"$NICname[$i]RCVframe,";
    print FILE "$NICname[$i]RCVcompressed,$NICname[$i]RCVmulticast,$NICname[$i]SNDbytes,";
    print FILE "$NICname[$i]SNDpackets,$NICname[$i]SNDerrs,$NICname[$i]SNDdrop,";
    print FILE "$NICname[$i]SNDfifo,$NICname[$i]SNDcolls,";
    print FILE "$NICname[$i]SNDcarrier,$NICname[$i]SNDcompressed,";
}
print FILE "LoadAvg1,LoadAvg5,LoadAvg15," ;


while ($check != 0){

######################## CPU Monitoring from /proc/stat ##############################

    open(STAT, $CPUFile) or die "Error oppening file $!\n";
```

```
    while (my $line= <STAT>) {

        if ($line=~ /^cpu\s+[0-9]+/){
            @cpu = split /\s+/, $line;
            shift @cpu;
        }

        if ($line =~ /cpu\d\s(.*)/ ){
            push (@Ncpu,$1);

        }

    }
    close STAT;

###################### MEmory Monitoring from /proc/meminfo ##########################

    open (MEM, $MemFile) or die "Error opening file  $!\n";

    while (my $line = <MEM>) {

        if ($line =~ /MemTot.*:\s+(\d+)\s+.*/){
            $totalmemory=$1;
            goto CONTINUE;
        }

        if ($line =~ /MemFree.*:\s+(\d+)\s+.*/ ){
            $freememory=$1;
            goto CONTINUE;
        }

        if ($line =~ /Buffers.*:\s+(\d+)\s+.*/ ){
            $buffer=$1;
            goto CONTINUE;
        }

        if ($line =~ /^Cached:\s+(\d+)\s+.*/ ){
            $cached=$1;
            goto CONTINUE;
        }

        if ($line =~ /SwapTotal.*:\s+(\d+)\s+.*/ ){
            $SwapTotal=$1;
            goto CONTINUE;
        }

        if ($line =~ /SwapFree.*:\s+(\d+)\s+.*/ ){
            $SwapFree=$1;
            goto CONTINUE;
        }
        CONTINUE:
    }
    close MEM;
    ######################## Networking Monitoring From /proc/net/stat ##################

    open (NIC, '/proc/net/dev') or die "Error opening file  $!\n";

    while (my $line = <NIC>) {

        if ($line =~ /([a-zA-Z]+.*\d*):\s*(.*)/ ){

            push (@NICdata,$2);

        }
    }

    close NIC;
```

```
    ######################## Reading Load Average of system ################################

    open (LOAD, '/proc/loadavg') or die "Error opening file  $!\n";

    while (my $line = <LOAD>) {

            if ($line =~ /(\d+\.\d+\s+\d+\.\d+\s+\d+\.\d+)\s+.*/ ){

                        $load=$1;

                    }
                }

        close LOAD;

    ######################## Writing Collected data in current cycle to OutputFile ########

    $systemtime=qx(echo \$(date +\%s\%M));
    chomp $systemtime;
    $check=qx(cat $checkfile);


    my $month=qx(date | cut -d ' ' -f 2);
    chomp $month;

    my $day=qx(date | cut -d ' ' -f 4);
    chomp $day;

    $row +=1;

    print FILE "\n" ;
    print FILE "$row,$month,$day,$systemtime,$totalmemory,$freememory,";
    print FILE "$buffer,$cached,$SwapTotal,$SwapFree," ;
    print FILE join(',',@cpu,);
    print FILE ",";

    for (my $i=0; $i < $CPUcount; $i++) {
        my @tempcpu = split /\s/ , $Ncpu[$i];
        print FILE join (',',@tempcpu,);
        print FILE ",";
    }
    splice(@Ncpu);

    for (my $i=0; $i < $NICcount; $i++) {
        my @tempnic = split /\s+/ , $NICdata[$i];
        print FILE  join (',',@tempnic,);
        print FILE",";

    }
    splice(@NICdata);

            my @tempload = split /\s+/ , $load;
        print FILE  join (',',@tempload,);
        print FILE",";



    Time::HiRes::sleep(0.5);

}

print FILE "\n";
close (FILE);
print "The output file is $outfile\n";
exit 0;


#####################################################
```

```
# subroutines


sub usage {

    print "Usage:\n";
    print "-h for help\n";
    print "-v for verbose (more output ) \n";
    print "-d for debug (even more output)\n";
}


sub verbose {
    print "VERBOSE: " . $_[0] if ( $VERBOSE or $DEBUG );
}


sub debug {
    print "DEBUG: " . $_[0] if ( $DEBUG );
}
```

B.1.2 Power.pl

Script Power.pl

```
#!/usr/bin/perl

# needed packages
use strict "vars";
use Getopt::Std;
use Time::HiRes qw(gettimeofday);

# Global Variables
my $VERBOSE = 0;
my $DEBUG = 0;

# commandline options

my $opt_string = "vdh";
getopts( "$opt_string", \my %opt ) or usage() and exit(1);

$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};

# my $server=$opt{'a'};
# print "$server\n";

if ( $opt{'h'} ) {
    usage();
    exit 0;
}
if ( $opt{'s'} ) {
            stop();
    exit 0;
}

# MAIN PART ####################################

my $server=qx(hostname);
chomp $server;

my $nametime=qx(echo \$(date +\%s\%M));
chomp $nametime;

my $checkfile="check.temp";
my $outfile ="Power_Record_$server-$nametime.csv";


my $check=0;

$check=qx(cat $checkfile);

my $param;
my $command="ipmitool -I open sensor get \"Power Meter\"\
\"Power Supply 1\" \"Power Supply 2\"";

my $row=0;


if ($check == 0){
    print " \nPOWER MONITORING IS NOT RUNNING !!\n ";
    exit 0;
}

open (FILE, ">>$outfile") or die "Error creating file $!\n";

print FILE "row,Day,SystemTime,PowerMeter,PowerSupply1,PowerSupply2\n";

print "\n***** Recording Power consumption of $server ********\n\n";
```

```
while ($check != 0){

    $check=qx(cat $checkfile);
    my $day=qx(date | cut -d ' ' -f 2,3,4);
    my $systemtime=qx(echo \$(date +\%s\%M));
    chomp $systemtime;
    chomp $day;

    $row++;
    my @temp;

    open (COMMAND, "$command|") or die "Error opening file  $!\n";

    while (my $line = <COMMAND>) {
        if ($line=~ /Sensor\sReading\s+:\s+(\d+).*/ ){
            push (@temp,$1);
        }
    }
    close COMMAND;
    print  FILE "$row,$day,$systemtime,";
    print  FILE join (',',@temp,);
    print  FILE "\n";
    sleep 1;
}

close (FILE);
print "The POWER output file is $outfile\n";
exit 0;
```

B.1.3 Analysis.pl

Script Analysis.pl

```
#!/usr/bin/perl

# our needed packages

use strict;
use Getopt::Std;
use Time::HiRes qw(gettimeofday);
use Math::BigFloat;
use Statistics::Descriptive;
use utf8;
use List::Util qw(sum);

$| = 1;

# Global Variables
my $VERBOSE = 0;
my $DEBUG = 0;
my $vm = 0;

# commandline options

my $opt_string = "vdhf:t:m:r";
getopts( "$opt_string", \my %opt ) or usage() and exit(1);

$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};
$vm = 1 if $opt{'r'};


my $file=$opt{'f'};
my $time=$opt{'t'};
my $type=$opt{'m'};

if ( $opt{'h'} ) {
    usage();
    exit 0;
}
if ( $opt{'s'} ) {
            stop();
    exit 0;
}

####################################
my $testnumber;
my $starttime;
my $stoptime;
my $check=0;
my $count=0;
my $row=0;

my $sumbuffer=0;
my $sumcach=0;
my @MemMean;

my $PrevIdle=0;
my $PrevNonIdle=0;
my $PrevTotal=0;
my $PrevIOW=0;
my $PrevIRQ=0;
my $PrevSIRQ=0;
my $PrevGuest=0;

my @CpuMean;
```

```
my @IOWMean;
my @IRQMean;
my @SIRQMean;
my @GuestMean;

my @LoadMean;

my $filename=$file;


open (TIME, $time) or die "Error opening TIME file  $!\n";

while (my $line = <TIME>) {
    if ($line=~ /([A-Z a-z]+\d+):/ ){
        $exp=$1;


        my $MemOutFile="Memory-AverageOfTests-$out-$type.csv";
        my $CpuOutFile="Cpu-AverageOfTests-$out-$type.csv";
        my $LoadOutFile="Load-AverageOfTests-$out-$type.csv";

        if ($vm == 1){
            $out=$out . $type;
            $type="vm";
        }

        my $AvgFile="Load-AverageOfAllExperiments_$type.csv";

        open (AVGALL, ">>$AvgFile") or die "AVAREAGE ALL FILE $!\n";

        my $checkfile="ck_$type";
        my $ck=qx(cat $checkfile);

        if ($ck==1){
            print AVGALL "Memory,,,,,,CPU,,,,,,,,,Load\n";
            print AVGALL "Experiment,Mean of Means,Standard Deviation of Means,";
            print AVGALL "Average of Change in Buffer,Average of Change in Cach";
            print AVGALL ",,Experiment,Mean of Means,Standard Deviation of Means,";
            print AVGALL "Average of IOWait,Average of Steal,Average of Irq,";
            print AVGALL "Average of Guest,,Experiment,Mean of Means,";
            print AVGALL "Standard Deviation of Means\n";
            qx(echo "0" > $checkfile);
        }

        open (OUTMEM, ">$MemOutFile")or die "Error opening MEM_OUT file  $!\n";
        print OUTMEM "Row,TestNumber,Mean of test,Standar Deviation of test,";
        print OUTMEM "Buffer change, Cach change\n";

        open (OUTCPU, ">$CpuOutFile")or die "Error opening CPU_OUT file  $!\n";
        print OUTCPU "Row,TestNumber,Mean of test,Standar Deviation of test,,
        print OUTCPU "iowait,,irq,,softirq,,Guest\n";

        open (OUTLOAD, ">$LoadOutFile")or die "Error opening Load_OUT file  $!\n";
        print OUTLOAD "Row,TestNumber,Mean of test,Standar Deviation of test,\n";

    }

    if ($line=~ /([A-Z a-z]+\d+)test\s+(\d+)\s+STARTED\s+at\s+(\d*)/ ){
        $exp=$1;
        $testnumber=$2;
        $starttime=$3;
        $check=0;


    }
    if ($line=~ /([A-Z a-z]+\d+)test\s+(\d+)\s+STOPPED\s+at\s+(\d*)/ ){
```

```
        $stoptime=$3;
        $check=1;

    }

    if ($check==1){


        my $linecount=0;
        my $firstbuffer=0;
        my $buffer=0;
        my $firstcach=0;
        my $cach=0;
        my @MemTestAvg;
        my $firstline=0;

        my @CpuTestAvg;
        my @IOWTestAvg;
        my @IRQTestAvg;
        my @SIRQTestAvg;
        my @GuestTestAvg;



        my @LoadTestAvg;



        open (MAIN, $file) or die "Error opening file  $!\n";

        while (my $line = <MAIN>) {
            if ($line=~ /\d+,[A-Z a-z]+,\d*:*\d*:*\d*,(\d+),(\d+),(\d+),(\d+),(\d+),\
            \d+,\d+,(\d+),(\d+),(\d+),(\d+),(\d+),(\d+),(\d+),(\d+),(\d+),(\d+),)\
            +(\d+.\d+),(\d+.\d+),(\d+.\d+),/ ){

                if ($1>$starttime && $1<$stoptime){

                    $linecount++;
                    $row++;

                    ###### MEMORY CALCULATIONS ########################

                    my $FreeMem=$3+$4+$5;
                    my $UsedMem=$2-$FreeMem;

                    $MemTestAvg[$linecount]=$UsedMem;


                    if ( $firstline==0){
                        $firstline=1;
                        $firstbuffer=$4;
                        $firstcach=$5;
                        }
                    $buffer=$4;
                    $cach=$5;

                    ####### CPU CALCULATIONS ###########################
                    my $Idle=$9+$10;
                    my $NonIdle=$6+$7+$8+$11+$12+$13;
                    my $Total=$Idle+$NonIdle;
                    my $IOW=$10;
                    my $IRQ=$11;
                    my $SIRQ=$12;
                    my $Guest=$14;


                    $PrevTotal=$PrevIdle+$PrevNonIdle;
```

```
                my $diffTotal=$Total-$PrevTotal;
                my $diffIdle=$Idle-$PrevIdle;
                my $diffIOW=$IOW-$PrevIOW;
                my $diffIRQ=$IRQ-$PrevIRQ;
                my $diffSIRQ=$SIRQ-$PrevSIRQ;
                my $diffGuest=$Guest-$PrevGuest;

                my $CpuUsage=100*(($diffTotal-$diffIdle)/$diffTotal);
                $CpuTestAvg[$linecount]=$CpuUsage;

                 my $IOWP=100*(($diffTotal-($diffTotal-$diffIOW))/$diffTotal);
                $IOWTestAvg[$linecount]=$IOWP;

                my $IRQP=100*(($diffTotal-($diffTotal-$diffIRQ))/$diffTotal);
                $IRQTestAvg[$linecount]=$IRQP;

                my $SIRQP=100*(($diffTotal-($diffTotal-$diffSIRQ))/$diffTotal);
                $SIRQTestAvg[$linecount]=$SIRQP;

                 my $GuestP=100*(($diffTotal-($diffTotal-$diffGuest))/$diffTotal);
                $GuestTestAvg[$linecount]=$GuestP;

                $PrevIdle=$Idle;
                $PrevNonIdle=$NonIdle;
                $PrevIOW=$IOW;
                $PrevIRQ=$IRQ;
                $PrevSIRQ=$SIRQ;
                $PrevGuest=$Guest;


                ####### SYSTEMLOAD CALCULATIONS ###################
                $LoadTestAvg[$linecount]=$17;

            }

        }

    }

$count++;

## MEMORY#########################################
$buffer=$buffer-$firstbuffer;
$cach=$cach-$firstcach;
$sumbuffer=$sumbuffer+$buffer;
$sumcach=$sumcach+$cach;

my $TestStat=Statistics::Descriptive::Full->new();

$TestStat->add_data(@MemTestAvg);
my $MemTestMean=$TestStat->mean();
my $MemTestSD=$TestStat->standard_deviation();

## CPU ############################################
$TestStat=Statistics::Descriptive::Full->new();
$TestStat->add_data(@CpuTestAvg);
my $CpuTestMean=$TestStat->mean();
my $CpuTestSD=$TestStat->standard_deviation();

$TestStat=Statistics::Descriptive::Full->new();
$TestStat->add_data(@IOWTestAvg);
my $IOWTestMean=$TestStat->mean();

$TestStat=Statistics::Descriptive::Full->new();
$TestStat->add_data(@IRQTestAvg);
my $IRQTestMean=$TestStat->mean();

$TestStat=Statistics::Descriptive::Full->new();
```

```
        $TestStat->add_data(@SIRQTestAvg);
        my $SIRQTestMean=$TestStat->mean();

        $TestStat=Statistics::Descriptive::Full->new();
        $TestStat->add_data(@GuestTestAvg);
        my $GuestTestMean=$TestStat->mean();


        ## LOAD ##########################################
        $TestStat=Statistics::Descriptive::Full->new();

        $TestStat->add_data(@LoadTestAvg);
        my $LoadTestMean=$TestStat->mean();
        my $LoadTestSD=$TestStat->standard_deviation();

        ## WRITE TO FILE #################################
        print OUTMEM "$count,$testnumber,$MemTestMean,$MemTestSD,$buffer,$cach\n";
        print OUTCPU "$count,$testnumber,$CpuTestMean,$CpuTestSD,,$IOWTestMean,,";
        print OUTCPU "$IRQTestMean,,$SIRQTestMean,,$GuestTestMean\n";
        print OUTLOAD "$count,$testnumber,$LoadTestMean,$LoadTestSD\n";
        print "******************** $count ***********************************\n";

        $MemMean[$count]=$MemTestMean;
        $CpuMean[$count]=$CpuTestMean;
        $IOWMean[$count]=$IOWTestMean;
        $IRQMean[$count]=$IRQTestMean;
        $SIRQMean[$count]=$SIRQTestMean;
        $GuestMean[$count]=$GuestTestMean;

        $LoadMean[$count]=$LoadTestMean;

        close MAIN;

    }


}
my $Stat=Statistics::Descriptive::Full->new();
$Stat->add_data(@MemMean);
my $MemMeanMean=$Stat->mean();
my $MemMeanSD=$Stat->standard_deviation();

### CPU ######################################

$Stat=Statistics::Descriptive::Full->new();
$Stat->add_data(@CpuMean);
my $CpuMeanMean=$Stat->mean();
my $CpuMeanSD=$Stat->standard_deviation();


$Stat=Statistics::Descriptive::Full->new();
$Stat->add_data(@IOWMean);
my $IOWMeanMean=$Stat->mean();

$Stat=Statistics::Descriptive::Full->new();
$Stat->add_data(@IRQMean);
my $IRQMeanMean=$Stat->mean();


$Stat=Statistics::Descriptive::Full->new();
$Stat->add_data(@SIRQMean);
my $SIRQMeanMean=$Stat->mean();


$Stat=Statistics::Descriptive::Full->new();
$Stat->add_data(@GuestMean);
my $GuestMeanMean=$Stat->mean();
```

```
### LOAD ##################################

$Stat=Statistics::Descriptive::Full->new();
$Stat->add_data(@LoadMean);
my $LoadMeanMean=$Stat->mean();
my $LoadMeanSD=$Stat->standard_deviation();


my $FAB=$sumbuffer/$count;
my $FAC=$sumcach/$count;


print AVGALL "$out,$MemMeanMean,$MemMeanSD,$FAB,$FAC,,$out,$CpuMeanMean,$CpuMeanSD,";
print AVGALL "$IOWMeanMean,$IRQMeanMean,$SIRQMeanMean,$GuestMeanMean,,,$out,";
print AVGALL "$LoadMeanMean,$LoadMeanSD\n";

close TIME;
close OUTMEM;
close OUTCPU;
close OUTLOAD;

exit 0;
```

B.2 Supporting Scripts

B.2.1 mac.sh

Script mac.sh

```
#!/bin/bash
LC_CTYPE=C
export OS_USERNAME=mohsen
export OS_PASSWORD=*****
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://controller:35357/v2.0

vmcsv="/var/lib/nova/instances/sriov/vm.csv"
check=0

while [ "$check" == "0" ];
 do

  MAC=00:60:2F

  for i in {1..3}
   do
    IFS= read -d '' -r -n 1 char < /dev/urandom
    MAC+=$(printf -- ':%02x\n' "'$char")
   done
  mac=$(printf '%s\n' "$MAC")

  check=1

  if [ -f $vmcsv ];
   then

    while IFS=, read instance sr_mac virtio_mac br pci
     do

      if [ "$mac" == "$sr_mac" ] || [ "$mac" == "$virtio_mac" ];
      then

      check=0;
      break
      break
     fi
   done <  $vmcsv
  fi

  hosts=$(nova hypervisor-list | grep '[0-9]' | awk '{print $4}')

  if [ "$hosts" != ""  ];
   then

    while read -r line2;
     do
      vms=$(virsh -c qemu+tcp://$line2/system list --all --name)

      if [ "$vms" != ""  ];
       then
        while read -r line3;
         do

          record=$(virsh -c qemu+tcp://$line2/system dumpxml $line3 | grep -i "$mac")

         if [ "$record" != "" ];
          then
          check=0;
          break
          break
        fi
```

```
   done<<<"$vms"
  fi
 done<<<"$hosts"
 fi
done

echo "$mac"
```

B.2.2 pci.sh

Script pci.sh

```
#!/bin/bash
export OS_USERNAME=mohsen
export OS_PASSWORD=*****
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://controller:35357/v2.0


if [ "$1" != "" ];
  then
  host=$1;
else
  host=$(hostname)
fi

pci_all=$(lspci | grep -i ethernet | grep -i  virtual\ function | awk '{print $1}')
vm_all=$(virsh list --all --name)


if [ "$pci_all" == "" ];
 then
  echo "NO VF";
else

  c1=0
  c2=0

  while IFS=:. read -r  bus slot func;
   do
    let c01=$c01+1
    check=0
    address="<address type='pci' domain='0x0000' bus='0x$bus' slot='$Slot' function='0x$func'/>"

    while read -r line_vm;
     do
      if [ "$vm_all" == "" ];
       then
        record=""
       else
      record=$(virsh dumpxml $line_vm | grep -i "$address")
     fi


     if [ "$record" != "" ];
      then
       check=1
       let c02=$c02+1
       break;
    fi

   done <<< "$vm_all"

 if [ "$check" == "0" ];
```

```
  then
   echo "$address"
   break;
 fi

 done <<< "$pci_all"

if [ "$c01" == "$c02" ];
then
echo "ALL ARE IN USE";
fi
fi
```

B.2.3 sriov.sh

Script sriov.sh

```
#!/bin/bash

export OS_USERNAME=mohsen
export OS_PASSWORD=******
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://controller:35357/v2.0

log="/var/lib/nova/instances/sriov/log.txt"
vmcsv="/var/lib/nova/instances/sriov/vm.csv"
macsh="/var/lib/nova/instances/sriov/mac.sh"
pcish="/var/lib/nova/instances/sriov/pci.sh"
dr="/var/lib/nova/instances/sriov/"
VC=0

if [ "$3" != "" ];
 then
  host=$3
else
  host=$(hostname)
fi


instance=$1

echo "START:Request for Host:[ $host ] Instance:[ $instance ] to  $2" >> $log

if [ "$2" == "attach" ];
   then
      if [ ! -f $vmcsv ];
         then
         touch $vmcsv;
      fi
      echo " SR-IOV: Attaching..." >> $log

      if [ "$(cat $vmcsv | grep -i $instance)" == "" ]
         then

         mac=$($macsh)
         address=$($pcish)

         echo " SR-IOV: Creatin XML with mac addres $mac and PCI addres $address" >> $log

         echo -e "<interface type='hostdev' managed='yes'>
         <mac address='$mac'/>
         <source>
         $address
         </source>
         </interface>" > $dr$instance.xml
```

```
        if [ -f $dr$instance.xml ];
           then
           echo " SR-IOV: XML created successfully at $dr$instance.xml " >> $log
        fi

        OUT=$(virsh -c qemu+tcp://$host/system attach-device $instance $dr$instance.xml)

        if [ "$OUT" == "Device attached successfully" ];
           then
           rm $dr$instance.xml;
           echo "$instance,$mac,NA,NA,$address" >> $vmcsv
           VOUT="NOT ATTACHED"
        fi


        echo "SR-IOV: $OUT  VIRTIO: $VOUT"
        echo -e " SR-IOV: $OUT\n VIRTIO: $VOUT" >> $log

     else

     echo "VM $instance ALREADY HAS SR-IOV - TRY REATTACH OR DETACH "
   fi

elif [ "$2" == "reattach" ];
   then

   echo " SR-IOV: Reataching .... " >> $log
   address=$($pcish)
   RC=0
   while IFS=, read instc sriov_mac virtio_mac br pci;
      do

      if [ "$instance" == "$instc" ];
         then
         RC=1
         mac=$sriov_mac
         old_address=$pci
         virtio=$virtio_mac
         virt_br=$br;
      fi

      done < $vmcsv
      if [ "$RC" == "1" ];
         then

         echo " SR-IOV: Creatin XML with mac addres $mac and PCI addres $address at Host : $host" >> $log

         echo -e "<interface type='hostdev' managed='yes'>
         <mac address='$mac'/>
         <source>
         $address
         </source>
         </interface>" > $dr$instance.xml

         if [ -f $dr$instance.xml ];
         then
         echo " SR-IOV: XML created successfully  " >> $log
         fi

         OUT=$(virsh -c qemu+tcp://$host/system attach-device $instance $dr$instance.xml)

         if [ "$OUT" == "Device attached successfully" ];
         then
         old="$instance,$mac,$virtio,$virt_br,\(.*\)"
         sed -i "/$old/d" $vmcsv
         echo "$instance,$mac,$virtio,$virt_br,$address" >> $vmcsv
         rm $dr$instance.xml;
```

```
         fi

         echo "$OUT"
         echo " SR-IOV: $OUT" >> $log
      else
         echo "NOTHING TO REATTACH!!"
         echo "SR-IOV: NOTHING TO REATTACH!!" >> $log
      fi

elif [ "$2" == "detach" ];
   then
   echo " SR-IOV: Detaching...." >> $log
   DC=0
   while IFS=, read instc sriov_mac virtio_mac br pci;
      do

      if [ "$instance" == "$instc" ];
         then
         DC=1
         mac=$sriov_mac
         address=$pci
      fi

   done < $vmcsv

   if [ "$DC" == "1" ];
   then
      echo " SR-IOV: Creating XML with mac addres $mac and PCI addres $address at Host: $host" >> $log

      echo -e "<interface type='hostdev' managed='yes'>
      <mac address='$mac'/>
      <source>
      $address
      </source>
      </interface>" > $dr$instance.xml

      if [ -f $dr$instance.xml ];
      then
         echo " SR-IOV: XML created successfully  " >> $log
      fi

      OUT=$(virsh -c qemu+tcp://$host/system detach-device $instance $dr$instance.xml)

      rm $dr$instance.xml

      echo "$OUT"
      echo " SR-IOV: $OUT" >> $log
   else
      echo "NOTHING TO DETACH!!"
      echo "SR-IOV: NOTHING TO DETACH!!" >> $log
   fi

else
   echo "Error in command : Enter Host instancename MacAddress attach/detac"
fi
```

Appendix C

Graphs

C.1 Single VM Experiments

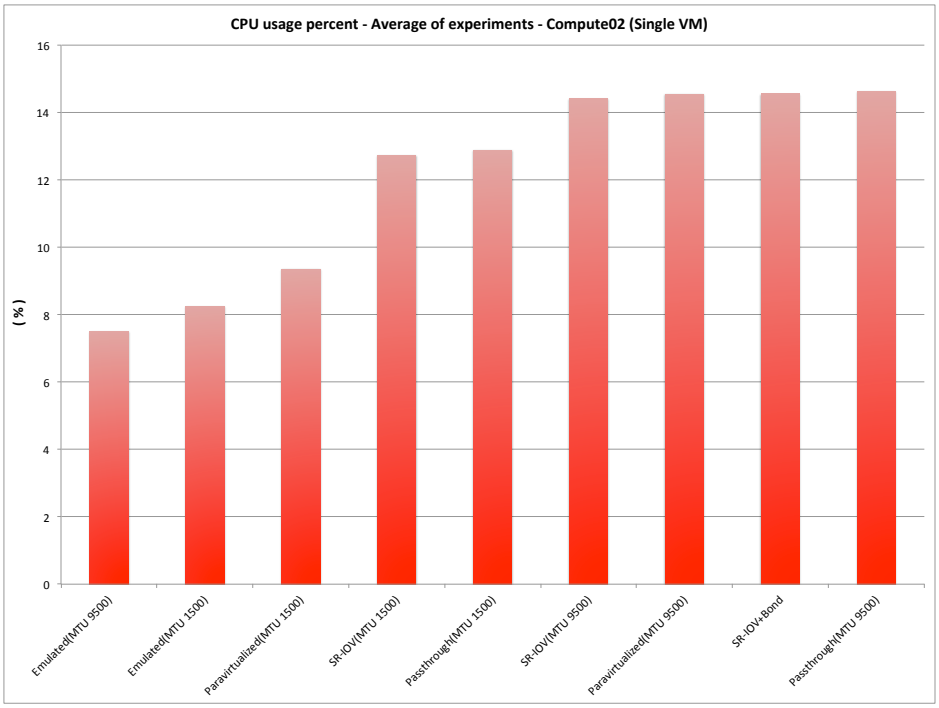


Figure C.1: CPU usage in Compute02- Average of experiments with single VM

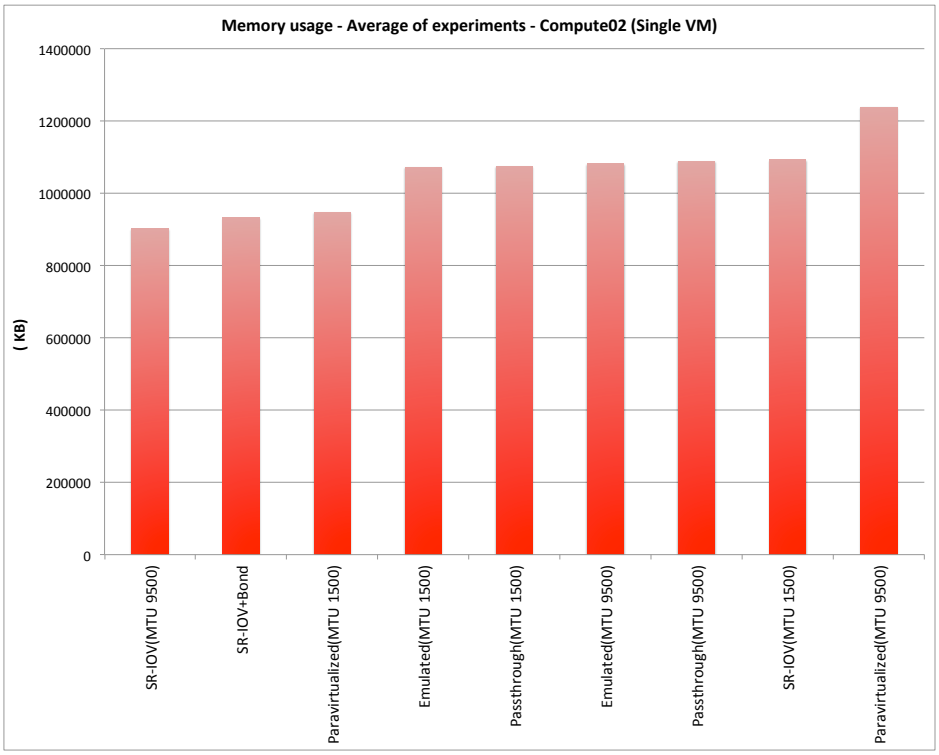


Figure C.2: Memory usage in Compute02- Average of experiments with single VM

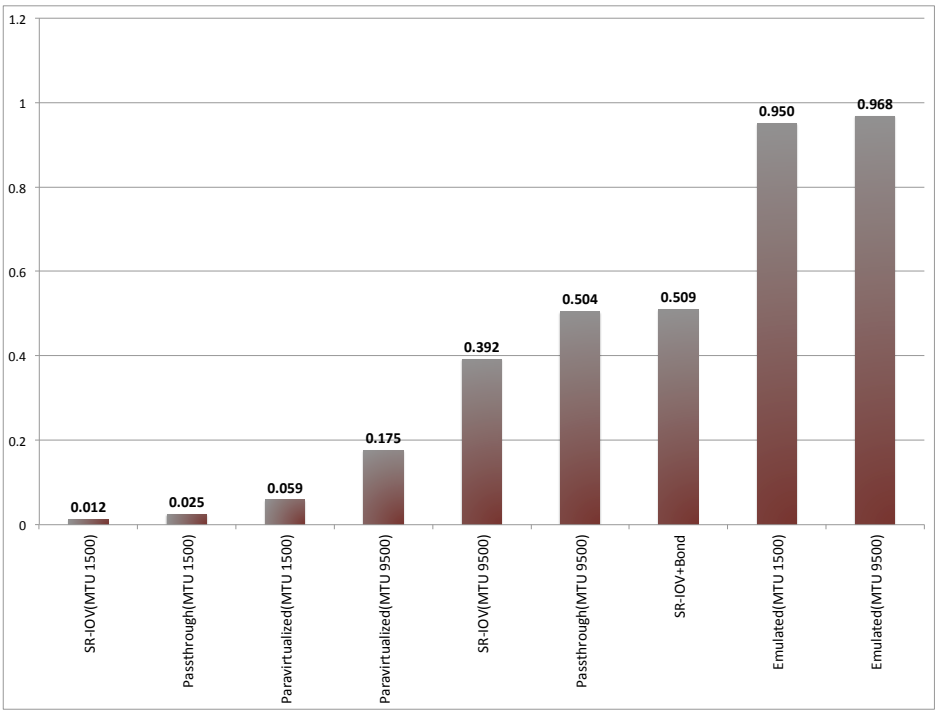


Figure C.3: Load average of different methods inside the VM during multiple VM experiments

C.2 Multiple VMs Experiments

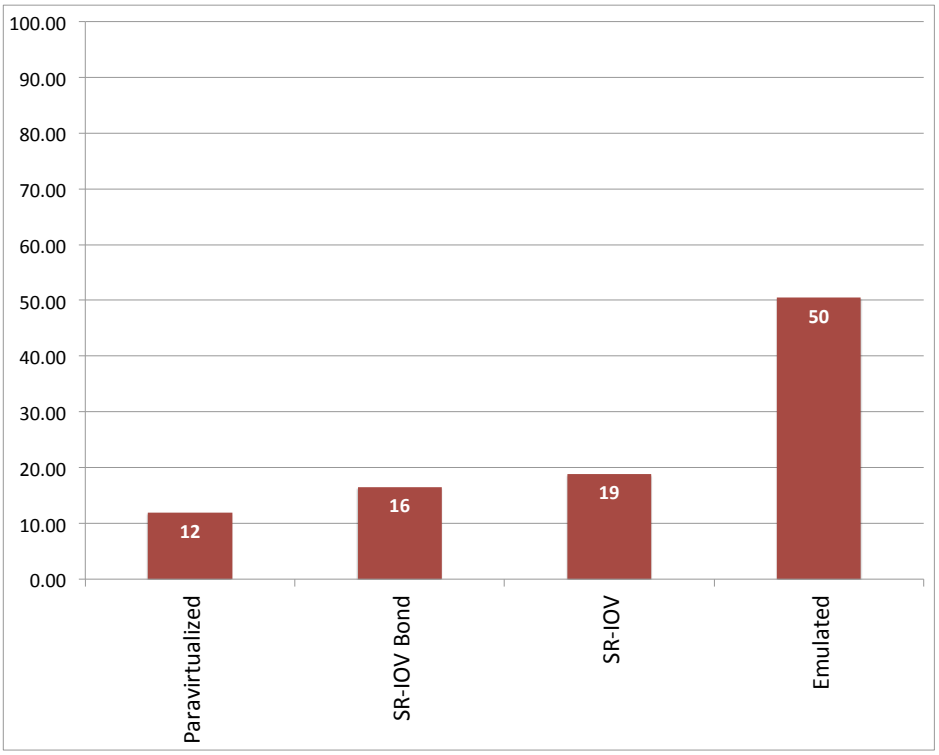


Figure C.4: Average of CPU usage inside the VM during multiple VM experiments

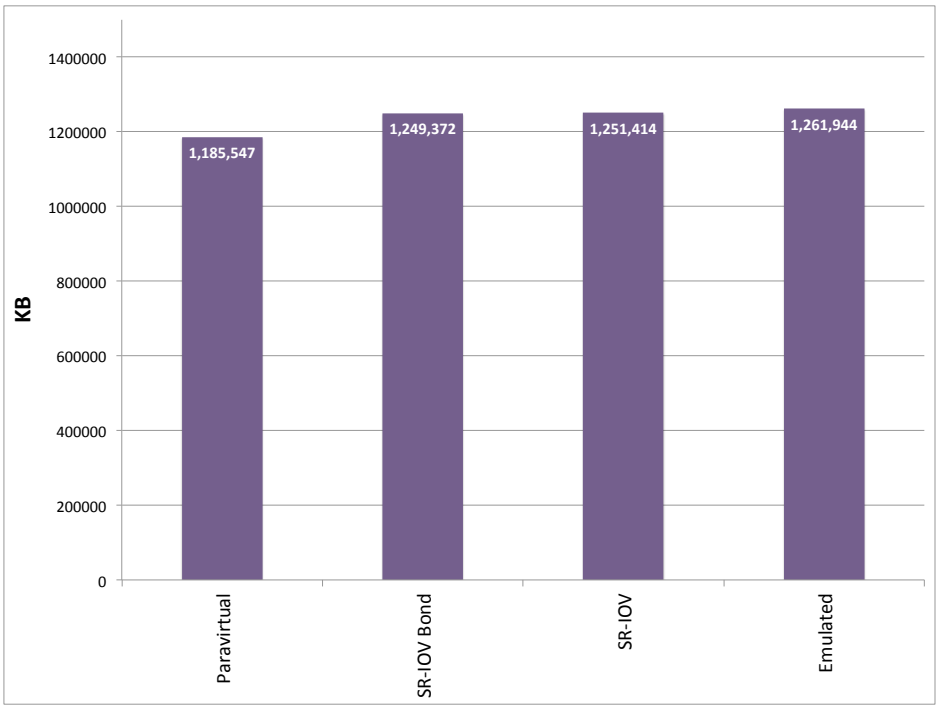


Figure C.5: Average of Memory usage in compute02 during multiple VM experiments

Bibliography

[1] Tim Abels, Puneet Dhawan and Balasubramanian Chandrasekaran. 'An overview of xen virtualization'. In: *Dell Power Solutions* 8 (2005), pp. 109–111.

[2] Padma Apparao et al. 'Characterization & analysis of a server consolidation benchmark'. In: *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM. 2008, pp. 21–30.

[3] Michael Armbrust et al. 'A View of Cloud Computing'. In: *Commun. ACM* 53.4 (Apr. 2010), pp. 50–58. ISSN: 0001-0782. DOI: 10.1145/1721654.1721672. URL: http://doi.acm.org/10.1145/1721654.1721672.

[4] S. Aust et al. 'Evaluation of Linux Bonding Features'. In: *Communication Technology, 2006. ICCT '06. International Conference on*. Nov. 2006, pp. 1–6. DOI: 10.1109/ICCT.2006.341935.

[5] Paul Barham et al. 'Xen and the art of virtualization'. In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), pp. 164–177.

[6] Andreas Berl et al. 'Energy-efficient cloud computing'. In: *The Computer Journal* 53.7 (2010), pp. 1045–1051.

[7] A Binu and G Santhosh Kumar. 'Virtualization techniques: a methodical review of XEN and KVM'. In: *Advances in Computing and Communications*. Springer, 2011, pp. 399–410.

[8] Intel IT Center. *Cloud Computing Research for IT Strategic Planning*. http://www.intel.com/content/www/us/en/cloud-computing/next-generation-cloud-networking-storage-peer-research-report.html?wapkw=peer+research. [Online; accessed 20-February-2014]. 2012.

[9] V. Chaudhary et al. 'A Comparison of Virtualization Technologies for HPC'. In: *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*. Mar. 2008, pp. 861–868. DOI: 10.1109/AINA.2008.45.

[10] Jianhua Che et al. 'A synthetical performance evaluation of OpenVZ, Xen and KVM'. In: *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*. IEEE. 2010, pp. 587–594.

[11] Wei Chen et al. 'A novel hardware assisted full virtualization technique'. In: *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*. IEEE. 2008, pp. 1292–1297.

[12] Christopher Clark et al. 'Live Migration of Virtual Machines'. In: *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286. URL: http://dl.acm.org/citation.cfm?id=1251203.1251223.

[13] CloudPassage. *Cloud Security Survey 2013: Private vs Public Cloud Deployment*. http://blog.cloudpassage.com/2013/12/13/2h-2013-cloud-security-survey-part-1-private-vs-public-cloud-deployment/. [Online; accessed 20-February-2014]. 2013.

[14] Jeff Daniels. 'Server virtualization architecture and implementation'. In: *Crossroads* 16.1 (2009), pp. 8–12.

[15] Marios D Dikaiakos et al. 'Cloud computing: distributed internet computing for IT and scientific research'. In: *Internet Computing, IEEE* 13.5 (2009), pp. 10–13.

[16] Tharam Dillon, Chen Wu and Elizabeth Chang. 'Cloud computing: issues and challenges'. In: *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. Ieee. 2010, pp. 27–33.

[17] Rogier Dittner and David Rule Jr. *The Best Damn Server Virtualization Book Period: Including Vmware, Xen, and Microsoft Virtual Server*. Syngress, 2011.

[18] Hanfei Dong et al. 'Formal Discussion on Relationship between Virtualization and Cloud Computing'. In: *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on*. IEEE. 2010, pp. 448–453.

[19] Yaozu Dong et al. 'High performance network virtualization with SR-IOV'. In: *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. Jan. 2010, pp. 1–10. DOI: 10.1109/HPCA.2010.5416637.

[20] Yaozu Dong et al. 'High performance network virtualization with SR-IOV'. In: *Journal of Parallel and Distributed Computing* 72.11 (2012), pp. 1471–1480.

[21] Yaozu Dong et al. 'Improving virtualization performance and scalability with advanced hardware accelerations'. In: *Workload Characterization (IISWC), 2010 IEEE International Symposium on*. Dec. 2010, pp. 1–10. DOI: 10.1109/IISWC.2010.5649499.

[22] Patrícia Takako Endo et al. 'A survey on open-source cloud computing solutions'. In: *Brazilian Symposium on Computer Networks and Distributed Systems*. 2010.

[23] J. Erbes, H.R. Motahari Nezhad and S. Graupner. 'The Future of Enterprise IT in the Cloud'. In: *Computer* 45.5 (May 2012), pp. 66–72. ISSN: 0018-9162. DOI: 10.1109/MC.2012.73.

[24] Tom Fitfield. 'Introduction to OpenStack'. In: *Linux J.* 2013.235 (Nov. 2013). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=2555789.2555793.

[25] I. Foster et al. 'Cloud Computing and Grid Computing 360-Degree Compared'. In: *Grid Computing Environments Workshop, 2008. GCE '08*. Nov. 2008, pp. 1–10. DOI: 10.1109/GCE.2008.4738445.

[26] Borko Furht. 'Cloud computing fundamentals'. In: *Handbook of cloud computing*. Springer, 2010, pp. 3–19.

[27] Patricia Gilfeather and Todd Underwood. 'Fragmentation and High Performance IP.' In: *IPDPS*. Citeseer. 2001, p. 165.

[28] GitHub. *iperf page at GitHub*. https://github.com/esnet/iperf. [Online; accessed 28-april-2014].

[29] Charles David Graziano. 'A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project'. In: (2011).

[30] Haibing Guan et al. 'SR-IOV Based Network Interrupt-Free Virtualization with Event Based Polling'. In: *Selected Areas in Communications, IEEE Journal on* 31.12 (Dec. 2013), pp. 2596–2609. ISSN: 0733-8716. DOI: 10.1109/JSAC.2013.131202.

[31] Qing Guo. *Methods and apparatus for measuring network performance*. US Patent 7,457,868. Nov. 2008.

[32] Irfan Habib. 'Virtualization with KVM'. In: *Linux J.* 2008.166 (Feb. 2008). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=1344209.1344217.

[33] Irfan Habib. 'Virtualization with kvm'. In: *Linux Journal* 2008.166 (2008), p. 8.

[34] Zhiqiang Huang et al. 'Adaptive and Scalable Optimizations for High Performance SR-IOV'. In: *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*. Sept. 2012, pp. 459–467. DOI: 10.1109/CLUSTER.2012.28.

[35] Michał Jankowski et al. 'Virtual Environments-Framework for Virtualized Resource Access in the Grid'. In: *Euro-Par 2006: Parallel Processing*. Springer, 2007, pp. 101–111.

[36] Linux journal. *Examining Load Average*. http://www.linuxjournal.com/article/9001. [Online; accessed 28-april-2014].

[37] Asim Kadav and Michael M Swift. 'Live migration of direct-access devices'. In: *ACM SIGOPS Operating Systems Review* 43.3 (2009), pp. 95–104.

[38] Linux Kernel. *official page*. https://www.kernel.org/doc/Documentation/networking/bonding.txt. [Online; accessed 28-april-2014].

[39] Avi Kivity et al. 'kvm: the Linux virtual machine monitor'. In: *Proceedings of the Linux Symposium*. Vol. 1. 2007, pp. 225–230.

[40] A. Kovari and P. Dukan. 'KVM amp; OpenVZ virtualization based IaaS open source cloud virtualization platforms: OpenNode, Proxmox VE'. In: *Intelligent Systems and Informatics (SISY), 2012 IEEE 10th Jubilee International Symposium on*. Sept. 2012, pp. 335–339. DOI: 10.1109/SISY.2012.6339540.

[41] G. von Laszewski et al. 'Comparison of Multiple Cloud Frameworks'. In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. June 2012, pp. 734–741. DOI: 10.1109/CLOUD.2012.104.

[42] Jean-Yves Le Boudec. *Performance evaluation of computer and communication systems*. EPFL Press, 2010.

[43] P.G.J. Leelipushpam and J. Sharmila. 'Live VM migration techniques in cloud environment 2014; A survey'. In: *Information Communication Technologies (ICT), 2013 IEEE Conference on*. Apr. 2013, pp. 408–413. DOI: 10.1109/CICT.2013.6558130.

[44] Ling Leng and Lin Wang. 'Research on cloud computing and key technologies'. In: *Computer Science and Information Processing (CSIP), 2012 International Conference on*. Aug. 2012, pp. 863–866. DOI: 10.1109/CSIP.2012.6308990.

[45] Peng Li. 'Centralized and Decentralized Lab Approaches Based on Different Virtualization Models'. In: *J. Comput. Sci. Coll.* 26.2 (Dec. 2010), pp. 263–269. ISSN: 1937-4771. URL: http://dl.acm.org/citation.cfm?id=1858583.1858625.

[46] linux-kvm.org. *KVM official home page*. http://www.linux-kvm.org/page/Main_Page. [Online; accessed 5-March-2014].

[47] linux-kvm.org. *KVM official page for download and installation*. http://www.linux-kvm.org/page/Downloads. [Online; accessed 5-March-2014].

[48] Jiuxing Liu et al. 'High Performance VMM-Bypass I/O in Virtual Machines.' In: *USENIX Annual Technical Conference, General Track*. 2006, pp. 29–42.

[49] David Marshall, Wade A Reynolds and Dave McCrory. *Advanced server virtualization: VMware and Microsoft platforms in the virtual data center*. CRC Press, 2006.

[50] Sean Marston et al. 'Cloud computing The business perspective'. In: *Decision Support Systems* 51.1 (2011), pp. 176–189. ISSN: 0167-9236. DOI: http://dx.doi.org/10.1016/j.dss.2010.12.006. URL: http://www.sciencedirect.com/science/article/pii/S0167923610002393.

[51] Miguel Masmano et al. 'Xtratum: a hypervisor for safety critical embedded systems'. In: *11th Real-Time Linux Workshop*. 2009.

[52] Peter Mell and Tim Grance. 'Effectively and securely using the cloud computing paradigm'. In: *NIST, Information Technology Lab* (2009).

[53] Peter Mell and Tim Grance. 'The NIST definition of cloud computing'. In: *National Institute of Standards and Technology* 53.6 (2009), p. 50.

[54] Jeffrey C Mogul and Lucian Popa. 'What we talk about when we talk about cloud network performance'. In: *ACM SIGCOMM Computer Communication Review* 42.5 (2012), pp. 44–48.

[55] *Openstack homepage*. https://www.openstack.org/. [Online; accessed 3-March-2014].

[56] Openstack.org. *History of Openstack*. http://docs.openstack.org/training-guides/content/module001-ch003-core-projects.html. [Online; accessed 26-February-2014]. 2013.

[57] Openstack.org. *KVM configuration of Openstack*. http://docs.openstack.org/havana/config-reference/content/kvm.html. [Online; accessed 5-March-2014]. 2013.

[58] Openstack.org. *Openstack Hypervisor Support Matrix*. https://wiki.openstack.org/wiki/HypervisorSupportMatrix. [Online; accessed 5-March-2014]. 2013.

[59] G. Pallis. 'Cloud Computing: The New Frontier of Internet Computing'. In: *Internet Computing, IEEE* 14.5 (Sept. 2010), pp. 70–73. ISSN: 1089-7801. DOI: 10.1109/MIC.2010.113.

[60] PCI-SIG. *official home page*. http://www.pcisig.com/home/. [Online; accessed 10-March-2014].

[61] PCI-SIG. *SR-IOV Specification*. http://www.pcisig.com/specifications/iov/single_root/. [Online; accessed 10-March-2014].

[62] Junjie Peng et al. 'Comparison of Several Cloud Computing Platforms'. In: *Information Science and Engineering (ISISE), 2009 Second International Symposium on*. Dec. 2009, pp. 23–27. DOI: 10.1109/ISISE.2009.94.

[63] Benjamin Quétier, Vincent Neri and Franck Cappello. 'Scalability comparison of four host virtualization tools'. In: *Journal of Grid Computing* 5.1 (2007), pp. 83–98.

[64] Rackspace. *Openstack Journy*. http://www.rackspace.com/blog/rcloud-is-your-cloud-the-openstack-journey/. [Online; accessed 26-February-2014]. 2012.

[65] Himanshu Raj and Karsten Schwan. 'High performance and scalable I/O virtualization via self-virtualized devices'. In: *Proceedings of the 16th international symposium on High performance distributed computing*. ACM. 2007, pp. 179–188.

[66] Edward Ray and Eugene Schultz. 'Virtualization security'. In: *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*. ACM. 2009, p. 42.

[67] Bhaskar Prasad Rimal, Eunmi Choi and Ian Lumb. 'A taxonomy and survey of cloud computing systems'. In: *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*. Ieee. 2009, pp. 44–51.

[68] Jeffrey Shafer. 'I/O virtualization bottlenecks in cloud computing today'. In: *Proceedings of the 2nd conference on I/O virtualization*. USENIX Association. 2010, pp. 5–5.

[69] Amit Singh. *An introduction to virtualization. In http://www. kernelthread. com/publications/virtualization*. 2004.

[70] J.E. Smith and R. Nair. 'The architecture of virtual machines'. In: *Computer* 38.5 (May 2005), pp. 32–38. ISSN: 0018-9162. DOI: 10.1109/MC.2005.173.

[71] Sourceforge. *IPMI TOOL*. http://sourceforge.net/projects/ipmitool/. [Online; accessed 28-april-2014].

[72] Open Stack. *HAVANA release*. https://www.openstack.org/software/havana/. [Online; accessed 28-april-2014].

[73] Jun Suzuki et al. 'Multi-Root Share of Single-Root I/O Virtualization (SR-IOV) Compliant PCI Express Device'. In: *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*. IEEE. 2010, pp. 25–31.

[74] A.S. Tanenbaum. *Computer Networks*. Computer Networks s. 3. Prentice Hall PTR, 2003. ISBN: 9780130661029. URL: http://books.google.no/books?id=Pd-z64SJRBAC.

[75] Tasoulas V, Haugerud Harek and Begnum K. 'Bayllocator: A Proactive System to Predict Server Utilization and Dynamically Allocate Memory Resources Using Bayesian Networks and Ballooning'. In: *Presented as part of the 26th Large Installation System Administration Conference*. 2012.

[76] S.J. Vaughan-Nichols. 'New Approach to Virtualization Is a Lightweight'. In: *Computer* 39.11 (Nov. 2006), pp. 12–14. ISSN: 0018-9162. DOI: 10.1109/MC.2006.393.

[77] Jeffrey Voas and Jia Zhang. 'Cloud computing: New wine or just a new bottle?' In: *IT professional* 11.2 (2009), pp. 15–17.

[78] Werner Vogels. 'Beyond server consolidation'. In: *Queue* 6.1 (2008), pp. 20–26.

[79] M.A. Vouk. 'Cloud computing: Issues, research and implementations'. In: *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*. June 2008, pp. 31–40. DOI: 10.1109/ITI.2008.4588381.

[80] Carl Waldspurger and Mendel Rosenblum. 'I/O Virtualization'. In: *Commun. ACM* 55.1 (Jan. 2012), pp. 66–73. ISSN: 0001-0782. DOI: 10.1145/2063176.2063194. URL: http://doi.acm.org/10.1145/2063176.2063194.

[81] Guohui Wang and T.S.E. Ng. 'The Impact of Virtualization on Network Performance of Amazon EC2 Data Center'. In: *INFOCOM, 2010 Proceedings IEEE*. Mar. 2010, pp. 1–9. DOI: 10.1109/INFCOM.2010.5461931.

[82] Lizhe Wang et al. 'Cloud computing: a perspective study'. In: *New Generation Computing* 28.2 (2010), pp. 137–146.

[83] Xiaolong Wen et al. 'Comparison of open-source cloud management platforms: OpenStack and OpenNebula'. In: *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*. May 2012, pp. 2457–2461. DOI: 10.1109/FSKD.2012.6234218.

[84] wiki.qemu.org. *QEMU official home page*. http://wiki.qemu.org/Main_Page. [Online; accessed 5-March-2014].

[85] Dong Xu. 'Cloud Computing: An emerging technology'. In: *Computer Design and Applications (ICCDA), 2010 International Conference on*. Vol. 1. June 2010, pages. DOI: 10.1109/ICCDA.2010.5541105.

[86] Xudong Xu et al. 'A VM migration and service network bandwidth analysis model in IaaS'. In: *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*. Apr. 2012, pp. 123–125. DOI: 10.1109/CECNet.2012.6201455.

[87] Xun Xu. 'From cloud computing to cloud manufacturing'. In: *Robotics and Computer-Integrated Manufacturing* 28.1 (2012), pp. 75–86. ISSN: 0736-5845. DOI: http://dx.doi.org/10.1016/j.rcim.2011.07.002. URL: http://www.sciencedirect.com/science/article/pii/S0736584511000949.

[88] Lamia Youseff, Maria Butrico and Dilma Da Silva. 'Toward a unified ontology of cloud computing'. In: *Grid Computing Environments Workshop, 2008. GCE'08*. IEEE. 2008, pp. 1–10.

[89] Edwin Zhai, Gregory D Cummings and Yaozu Dong. 'Live migration with pass through device for Linux VM'. In: *OLS08: The 2008 Ottawa Linux Symposium*. 2008, pp. 261–268.

[90] Binbin Zhang et al. 'Evaluating and optimizing I/O virtualization in kernel-based virtual machine (KVM)'. In: *Network and Parallel Computing*. Springer, 2010, pp. 220–231.

[91] Qi Zhang, Lu Cheng and Raouf Boutaba. 'Cloud computing: state-of-the-art and research challenges'. In: *Journal of internet services and applications* 1.1 (2010), pp. 7–18.